Propriété littéraire et scientifique réservée pour tous les pays du monde. Ce document ne peut être reproduit ou traduit en tout ou en partie sans l’autorisation écrite du Directeur général du CERN, titulaire du droit d’auteur. Dans les cas appropriés, et s’il s’agit d’utiliser le document à des fins non commerciales, cette autorisation sera volontiers accordée.

Le CERN ne revendique pas la propriété des inventions brevettées et dessins ou modèles susceptibles de dépôt qui pourraient être décrits dans le présent document; ceux-ci peuvent être librement utilisés par les instituts de recherche, les industriels et autres intéressés. Cependant, le CERN se réserve le droit de s’opposer à toute revendication qu’un usager pourrait faire de la propriété scientifique ou industrielle de toute invention et tout dessin ou modèle décrits dans le présent document.

© Copyright CERN, Genève, 1972

Literary and scientific copyrights reserved in all countries of the world. This report, or any part of it, may not be reprinted or translated without written permission of the copyright holder, the Director-General of CERN. However, permission will be freely granted for appropriate non-commercial use. If any patentable invention or registrable design is described in the report, CERN makes no claim to property rights in it but offers it for the free use of research institutions, manufacturers and others. CERN, however, may oppose any attempt by a user to claim any proprietary or patent rights in such inventions or designs as may be described in the present document.
ORGANIZING COMMITTEE

Professor D.W. Barron (Southampton)
Professor J.J. Duby (Paris)
Dr. D. Kuhn (Innsbruck)
Dr. W.O. Lock (CERN)
Dr. G.R. Macleod (CERN)
Professor H. Pietschmann (Vienna)
Dr. K. Runge (CERN)
Dr. P. Zanella (CERN)

Mrs. I. Barnett (CERN)
Miss D.A. Caton (CERN)
PREFACE TO THE PROCEEDINGS OF THE
1972 CERN COMPUTING AND DATA PROCESSING SCHOOL

The second CERN Computing and Data Processing School was held at Pertisau in the Austrian Tyrol from 10-24 September 1972, and it is a pleasure to thank our hosts -- the Austrian Academy of Sciences and the Tyrol Regional Government -- for their hospitality. Much of the burden of the local arrangements fell upon the the Director of the School, Professor H. Pietschmann of the Institute for Theoretical Physics of the University of Vienna, and on Dr. D. Kuhn of the Physics Institute of the University of Innsbruck; we are most grateful to them for their invaluable contribution.

The programme of the School covered topics in computing in experimental high-energy physics, computer science and applied mathematics, and the lectures were generally of a high standard. The discussions generated outside the lecture room and the personal contacts made between people working on similar topics in different laboratories are also an important aspect of these Schools, and the Pertisau School proved to be a very lively and stimulating experience.

I would like to express, on behalf of my colleagues on the Organizing Committee, our thanks to the lecturers for their work, to the students at the School for their active participation, and to Miss D.A. Caton and Mrs. I. Barnett of the CERN Conference Secretariat for their efficient organization of the School. My only regret is that the Organizing Committee was unable to arrange for more clement weather during the School.

The present Proceedings are a photo-offset reproduction of the manuscripts provided by the lecturers, and I would like to thank the CERN Document Reproduction Service for their work in publishing the Proceedings so soon after the School.

G.R. Macleod
Chairman
Organizing Committee
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION MINIMIZATION</td>
<td>1</td>
</tr>
<tr>
<td>F. James</td>
<td></td>
</tr>
<tr>
<td>FUNCTION PARAMETRIZATION</td>
<td>53</td>
</tr>
<tr>
<td>H. Wind</td>
<td></td>
</tr>
<tr>
<td>SMALL COMPUTERS IN PHYSICS</td>
<td>107</td>
</tr>
<tr>
<td>I.C. Pyle</td>
<td></td>
</tr>
<tr>
<td>SYSTEM ASPECTS OF SMALL COMPUTERS IN PARTICLE PHYSICS: A PERSONAL VIEW</td>
<td>187</td>
</tr>
<tr>
<td>B. Zacharov</td>
<td></td>
</tr>
<tr>
<td>THE OMEGA PROJECT: A CASE HISTORY IN THE DESIGN AND</td>
<td>275</td>
</tr>
<tr>
<td>IMPLEMENTATION OF AN ON-LINE DATA ACQUISITION SYSTEM</td>
<td></td>
</tr>
<tr>
<td>R.D. Russell</td>
<td></td>
</tr>
<tr>
<td>ON BOARD COMPUTERS FOR SATELLITES</td>
<td>341</td>
</tr>
<tr>
<td>M.A. Perry</td>
<td></td>
</tr>
<tr>
<td>BIRTH AND LIFE OF THE BOL-SYSTEM</td>
<td>365</td>
</tr>
<tr>
<td>J.E.J. Oberski</td>
<td></td>
</tr>
<tr>
<td>RADIO TELESCOPE CONTROL</td>
<td>385</td>
</tr>
<tr>
<td>J. Schraml</td>
<td></td>
</tr>
<tr>
<td>CASE HISTORY: A HIGH-ENERGY PHYSICS EXPERIMENT</td>
<td>407</td>
</tr>
<tr>
<td>S. Conetti</td>
<td></td>
</tr>
<tr>
<td>CONTENTS (cont.)</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>ACCELERATOR CONTROL</td>
<td>423</td>
</tr>
<tr>
<td>R. Keyser</td>
<td></td>
</tr>
<tr>
<td>SIGNIFICANT CONCEPTS IN DATA BASE MANAGEMENT</td>
<td>443</td>
</tr>
<tr>
<td>T.W. Olle</td>
<td></td>
</tr>
<tr>
<td>THEORY AND PRACTICE OF COMPILATION</td>
<td>449</td>
</tr>
<tr>
<td>H. Langmaack</td>
<td></td>
</tr>
<tr>
<td>LIST OF PARTICIPANTS</td>
<td>497</td>
</tr>
</tbody>
</table>
FUNCTION MINIMIZATION

F. James
CERN, Geneva, Switzerland

1. INTRODUCTION

1.1 The motivation

A large class of problems in many different fields of research can be reduced to the problem of finding the smallest value taken on by a function of one or more variable parameters. Examples come from fields as far apart as industrial processing (minimization of production costs) and general relativity (determination of geodesics by minimizing the path length between two points in curved space-time). But the classic example which occurs so often in scientific research is the estimating of unknown parameters in a theory by minimizing the difference (chi-square) between theory and experimental data. In all these examples, the function to be minimized is of course determined by considerations proper to the particular field being investigated, which will not concern us in these lectures. Our aim is to study the mathematical problem of minimization.

1.2 Minimization, maximization, and optimization

Although traditionally one speaks of function minimization, some authors refer to maximization. Of course the two are entirely equivalent since one can be converted to the other by changing the sign of the function. Thus the problems of minimizing chi-square, maximizing likelihood, minimizing cost, or maximizing efficiency can all be considered as minimization (or maximization). To avoid committing oneself, it is now fashionable to speak of optimization, to cover both cases. This unfortunately causes confusion with optimization in control theory where the principal techniques are analytical (calculus of variations) and hence bear little relationship to the numerical methods used in function minimization as treated here.
To add to the confusion there is the term "programming", which is also used to mean minimization (usually specified as linear programming, non-linear programming, or mathematical programming), a historical usage dating from the time when programmers in the modern sense did not exist, and computer users were not programming but coding.

Other terms used for minimization are extremization and hill-climbing. Since these can also be used to mean other things, the general conclusion is that in this field you can not tell a book from its title. While waiting for general agreement as to what the subject should be called, I will stick to function minimization.

1.3 Definition of the problem

Given a function $F(x)$, the general problem is to find the value of the variable or variables $x$ for which the function $F(x)$ takes on its smallest value. [As pointed out above, this is entirely equivalent to finding the $x$ for which the function $-F(x)$ takes on its largest value, but for consistency we will always consider only minimization.] The rules of the game are the following:

i) The function $F(x)$ is assumed not to be known analytically, but is specified by giving its value at any point $x$.

ii) The allowed values of the variable or variables $x$ may be restricted to a certain range, in which case one speaks of constrained minimization. In these lectures we limit ourselves to the unconstrained problem.

iii) In some cases additional information about the function $F$ may be available, such as the numerical values of the derivatives $\partial F/\partial x$ at any point $x$. Such knowledge cannot in general be assumed, but should be used when possible.

iv) The function $F(x)$ is repeatedly evaluated at different points $x$ until its minimum value is attained.

The method which finds the minimum (within a given tolerance) after the fewest function evaluations is the winner. Occasionally other considerations may be important, such as the amount of storage required by the
method or the amount of computation required to implement the method, but normally the dominating factor will be the time spent in evaluating the function.

1.4 Definition of a minimum

The theorems of elementary calculus tell us that the function $F(x)$ must take on its smallest value at a point where either:

i) all derivatives $\frac{\partial F}{\partial x} = 0$ (a stationary point), or

ii) some derivative $\frac{\partial F}{\partial x}$ does not exist (a cusp), or

iii) the point $x$ is on the boundary of the allowed region (an edge point).

Although we will sometimes find it useful to consider points satisfying the above properties, this approach of considering essentially the analytic properties of the function is clearly not well adapted to the rules of the game as outlined above. Indeed, when one considers that there may be any number of stationary points, cusps, and edge points, all of which may be arbitrarily hard to find by simply sampling the function value, the whole problem begins to appear hopeless unless some simplifying assumptions are made.

The usual simplification consists in abandoning the attempt to find the global minimum and being satisfied with a local minimum. A local minimum may be defined as a point $x_0$, where for all points $x$ in some neighbourhood around $x_0$ we have $F(x) > F(x_0)$.

Now the situation looks much brighter since the very definition of a local minimum suggests a general strategy for finding one: we vary $x$ by small steps in a direction which causes $F$ to decrease, and continue until $F$ increases in all allowed directions from some point $x_0$. This does not yet tell us how to vary $x$, but at least it suggests that a solution can be found.

In the lectures we will consider only unconstrained local minimization, unless otherwise stated. The problem of global minimization will be treated in Section 6.
1.5 The shape of the function -- Taylor's series

With a view to making an intelligent minimizing method, it is of interest to consider what we might reasonably expect about the behaviour of $F$. If $F$ represents a physically meaningful function, we would certainly expect all the derivatives of $F$ to exist everywhere in the region of interest. Under these conditions we can write down the Taylor's series expansion for $F$ about some point $x_1$, assuming for the moment that $x$ represents just one variable:

$$F(x) = F(x_1) + \frac{\partial F}{\partial x} \bigg|_{x_1} (x-x_1) + \frac{1}{2} \frac{\partial^2 F}{\partial x^2} \bigg|_{x_1} (x-x_1)^2 + \ldots$$

Although we do not know anything a priori about the domain of convergence of this series, we do know that as the distance $(x-x_1)$ becomes smaller, the higher order terms become less important, so that we would expect that predictions based on the low-order terms should not be very wrong, at least for small steps. Before considering these terms in more detail, let us generalize the variable $x$ to a vector of $n$ variables $\underline{x}$. Then we have

$$F(\underline{x}) = F(\underline{x}_1) + \underline{g}^T(\underline{x}-\underline{x}_1) + \frac{1}{2} (\underline{x}-\underline{x}_1)^T \underline{G}(\underline{x}-\underline{x}_1) + \ldots,$$

where the matrix $\underline{G}$ is defined by $G_{ij} = \frac{\partial^2 F}{\partial x_i \partial x_j}$, and the gradient vector $\underline{g}$ is $g_i = \frac{\partial F}{\partial x_i}$, all derivatives being evaluated at $\underline{x}_1$. The $T$ denotes transposition which turns a column vector into a row vector. Note the difference between $x_i$ (the $i^{th}$ variable) and $\underline{x}_i$ (the position vector at the point $i$).

Now the first term of the above series is constant, so it will not tell us much about where to look for a minimum. The second term is proportional to the gradient $\underline{g}$ and tells us in which direction the function is decreasing the fastest, but since it is linear in $\underline{x}$, it does not predict a minimum and therefore does not tell us what step size to take. Moreover, as we approach the minimum $\underline{g} \rightarrow 0$ (if it exists) so we will have to go further and consider the next term. The third, or quadratic term describes a parabolic behaviour and is therefore the lowest term to
predict a minimum. Unlike \( g \) we can expect \( \zeta \) to be roughly constant over small regions, since it would be exactly constant if higher-order terms were zero.

We mention, in passing, one class of problems in which the above analysis would not hold at all. This is in the field known as linear programming, which limits itself to minimizing functions which are linear in the parameters, subject to constraints which are also linear. A linear function can not have a minimum in the sense described above (a stationary point) but must take on its minimum at a constraint boundary (edge point). For such problems the description of the constraints therefore takes on greater importance than the analysis of the function itself, and will not be considered in these lectures.

1.6 Non-existence of optimum in general

Although much of these lectures will be spent studying and comparing different minimization algorithms (recipes), the reader should be warned at the outset that in the strict sense of the rules of the game as stated in Section 1.3 above, we will not be able to show any algorithm to be superior to any other for all functions. In principle at least, no matter how bad one algorithm is, or how good another, we can always find a function which will be minimized faster by the bad method than by the good one. We should keep such essentially theoretical considerations in mind, but should not be overly discouraged by them. In particular, certain objective criteria will emerge for comparing methods even though the principal criterion -- minimization speed -- depends on the function. In the past there has in my opinion been an overemphasis on such objective criteria in an attempt to find the ideal universal minimization algorithm. More recently, the tendency is to adapt the algorithm to the function, even to the point of introducing a super-algorithm which would choose a sub-algorithm appropriate to the function at hand. Such questions of global strategy will be considered in the last lecture.

The reader should also be warned that in presenting particular algorithms I will often omit details which are unimportant to an understanding of the algorithm although they may be crucial in actually making it work. The original references should therefore be consulted before programming such algorithms.
1.7 The role of the computer

While our subject is essentially a mathematical one, it has been so profoundly influenced by the existence of high-speed electronic computers that it would certainly be unfair not to mention them here. Indeed, real progress in the solving of large-scale practical problems has come only in the last fifteen years, although much of the basic theory dates back to Newton's time or even earlier. This is, of course, because of the renewed interest in numerical minimization techniques for use on computers. As it is no longer even thinkable to use these techniques for hand calculations, it is best to place ourselves immediately in the computer context and to conceive of our function \( F(x) \) rather as a subroutine which returns a value of \( F \) (and perhaps some other information such as numerical values of derivatives) for given input values of the arguments \( x \).

One unpleasant consequence of the computer-oriented approach is that we will sometimes have to worry about rounding-off errors in the function value due to the finite word length of digital computers. This rounding error is usually of the order of \( 10^{-8} \) of the function value for computers with 36-bit words in single precision, but the cumulative effects of rounding inside a complicated function subroutine may be much larger. In addition there may be problems of overflow or underflow.

2. ONE-DIMENSIONAL MINIMIZATION

2.1 Usefulness in \( n \)-dimensional problems

We will first consider functions of just one variable, since some general problems can be seen more easily in this simplest case and also because some \( n \)-variable algorithms contain steps which require one-dimensional minimization. The one-variable problem is therefore both instructive and useful even though our prime consideration will be that of more complex problems.
2.2 Grid search

The most elementary search technique consists in choosing k equally spaced points within the range of the parameter $x$, evaluating the function at each of the points, and retaining the lowest value found. If the spacing between points is $\Delta x$, one of the points is sure to be within $\Delta x/2$ of the true minimum, although in principle it may not be the point corresponding to the lowest value. Still, if the function does not vary too wildly over the distances of the order of $\Delta x$, one generally assumes that this method gives the minimum within a range of about $\Delta x$.

Of course the grid search method has some difficulties. It is not directly applicable to the usual case where the range of $x$ is infinite. But in this case a simple remedy is to choose a reasonable range in the middle of the allowed range, and later to shift the sampling range if the minimum comes out at an end point.

The most serious objection to the grid method is its inefficiency. Given the assumption that $F$ does not vary too much over a distance of $\Delta x$, many of the function evaluations are certainly unnecessary, namely those that are in regions where the function value is known to be large. In other words, the algorithm takes no account of what it has learned about the function. This inefficiency becomes more striking, in fact prohibitive, when extended to many variables.

On the other hand, this method has the prized virtues of extreme simplicity and absolute stability. It always converges within the desired tolerance in a known number of steps and is quite insensitive to the detailed behaviour of the function.

The efficiency of the grid method may be greatly improved by proceeding in several stages, using a smaller range and smaller step size in each succeeding stage. In this way each stage takes account of the least value found in the preceding stage, and the method can be said to converge in the usual sense of increasing accuracy due to decreasing step size. In the next section we consider optimum ways to arrange this staging in order to obtain the fastest decrease in step size.
2.3 Fibonacci and golden section searches

In order to optimize the grid search, we want to minimize the number of function evaluations per stage, compatible with maintaining a constant reduction of a factor $t$ in the step sizes at each stage. This will yield the fastest reduction in step size. One function evaluation tells us nothing about the possible location of a minimum, but as long as we restrict ourselves to local minima in a given range of $x$, two points are sufficient as shown in the diagram below. If $F(x_1) < F(x_2)$, then there must be at least one local minimum somewhere in the range $0 \leq x < x_2$. Now in this new range, we already have one point ($x_1$), so that a further reduction in range is possible with only one new function evaluation, and the procedure can now be continued with only one new evaluation per stage. It remains to be shown that this can be continued indefinitely with a constant reduction in step size, and to calculate what that reduction will be. Clearly we would get the maximum reduction on the first step if $x_1$ and $x_2$ were very close together, but we must not forget that $x_1$ (or $x_2$) will then be used for the next stage and should therefore be close to the middle of this new interval as well. The situation is illustrated in the diagram below, where the distances indicated are imposed by the symmetry of the intervals and the condition that the reduction in range must be a factor of $t$ in each stage. The new range after evaluation of $F(x_3)$ will be $x_3 < x < x_2$ and its length must be $t^2$. 
This will be possible since there is a real root to the equation:

\[ t^2 = 1 - t \]
\[ t = \frac{\sqrt{5} - 1}{2} \approx 0.616. \]

Since this ratio \( t \) is known as the golden section, the minimization technique is called a golden section search. If the number of stages to be taken is known in advance, it is possible to improve very slightly on this technique by using a Fibonacci search, as described for example in Kowalik and Osborne\(^1\). Although Fibonacci can be shown to be optimal (in a sense described below), the slight improvement is probably not worth the added complication. The golden section search is optimal among algorithms where the stopping point is not decided in advance.

The above techniques are optimal only in the minimax sense, that is, they minimize the maximum number of function evaluations necessary to obtain a given accuracy. It might be called the pessimists optimality, since in game theory it is the best strategy against an intelligent opponent who is trying to make you lose. It should therefore be effective in minimizing pathological functions, but in more normal cases we should expect other methods to be better. Such methods are described in the following sections.

2.4 Quadratic interpolation and extrapolation

A more optimistic approach consists in studying the expected behaviour of the function and then hoping that the deviations of the real function from this behaviour are not too great. From the Taylor’s series analysis of Section 1.5, it would be reasonable to proceed by assuming that the function is nearly quadratic.

Since a parabola is determined by three points, this method requires the function to have been evaluated for three different values \( x_1, x_2, \) and \( x_3 \). It then predicts the minimum to be at the minimum of the parabola passing through these points. If the three function values are \( F_1, F_2, \) and \( F_3 \), the predicted minimum is at \( x_4 \), given by
\[
x_4 = \frac{(x_2 + x_3)F_1}{(x_1 - x_2)(x_1 - x_3)} + \frac{(x_1 + x_3)F_2}{(x_2 - x_1)(x_2 - x_3)} + \frac{(x_1 + x_2)F_3}{(x_3 - x_1)(x_3 - x_2)}
\]

Considerable simplification results when the three points are equally spaced, a distance \(d\) apart, in which case

\[
x_4 = x_2 + \frac{d}{2} \frac{(F_1 - F_3)}{(F_1 + F_3 - 2F_2)}.
\]

The function is then evaluated at \(x_4\), this point replaces one of the first three, and a new point is predicted, again by quadratic interpolation using the new set of three points. The method terminates when the predicted function value at some new point agrees with the actual value within a specified tolerance.

This algorithm usually performs quite well when applied to easy (nearly quadratic) functions, but suffers from a number of instabilities which can be quite serious, as follows:

i) At any step the three points may determine a parabola with a maximum rather than a minimum, in which case the method diverges.

ii) If the three points lie nearly in a straight line, the algorithm takes an enormous step which may cause numerical difficulties as well as diverging.

iii) After each step there is a choice of which two of the three previous points to retain for the next step. It is usually more convenient and logical to retain the most recent points, but this may also lead to instabilities by throwing away the best points.

iv) Even without any of the above difficulties, the method may oscillate about the minimum instead of converging toward it.

All the problems can be fixed by including checks and safeguards in the algorithm, but the remedies always involve abandoning, at least temporarily, the quadratic interpolation step. The best remedy is probably
to reserve the method for well-behaved functions and to abandon it entirely as soon as trouble arises. It is most often used as the last step in algorithms which depend principally on other methods, since physical functions are usually quite parabolic in the immediate vicinity of the minimum.

When derivatives of the function are available, variations of quadratic interpolation are possible, using instead of three points to determine the parabola, either two function values and one first derivative, or the function value and the first two derivatives at one point. These variations tend to be even more unstable than the basic method, since they use information from fewer points.

2.5 The success-failure method

A good compromise between the stability of the grid search and the rapid convergence of quadratic interpolation is found with the success-failure technique of Rosenbrock\(^2\). A start point \(x_0\) and initial step size \(d\) are required, and the function is evaluated at \(x_0\) and \(x_0 + d\). The first step is termed a success if \(F(x_0 + d) < F(x_0)\), otherwise it is a failure. If it is a failure, \(d\) is replaced by \(-\beta d\), where \(\beta\) is a contraction factor less than one, and the test is repeated. If it is a success, \(x_0\) is replaced by \(x_0 + d\), \(d\) is replaced by \(\alpha d\), where \(\alpha\) is an expansion factor greater than one, and the test is repeated. The process continues in this way until the function values change by less than a specified amount. The numerical values usually used for the expansion and contraction parameters are \(\alpha \approx 3.0\) and \(\beta \approx 0.4\).

An interesting feature of this method is that a local minimum is always bracketed whenever a success is followed by a failure. When this happens, the middle one of the last three points is always lower than the outer two, so that one is in a favourable position for trying a quadratic interpolation step. The success-failure method, with one quadratic interpolation step each time a success is followed by a failure, is probably the most effective one-dimensional technique for use on general functions, although in special cases other methods may be superior.
3. **STEPPING METHODS IN MANY VARIABLES**

3.1 Grid searches and random searches

An excellent illustration of the enormous increase in complexity in going to spaces of high dimensionality is afforded by the grid search technique in many variables. In order to localize a minimum to 1% of the range of one variable by this technique requires 100 function evaluations; in ten variables the number of points required is $10^{20}$. Clearly we can forget about this method when more than one or two parameters are involved.

In fact it is a general rule in function minimization, as in function integration, that one should not expect good one-dimensional techniques to be good when extended to higher dimensionality. Experience with integration suggests that a Monte Carlo search is more efficient than a grid search in many dimensions. The Monte Carlo technique consists in choosing points randomly according to some distribution (usually uniform or normal).

But even when these methods are refined by using variable search ranges, they prove far too slow for general use and we must turn to more efficient techniques.

3.2 Single-parameter variation

Since the condition for a minimum which is a stationary point in $n$ variables $x_i$ is the vanishing of all $n$ first derivatives $\partial F/\partial x_i$, it is natural to try to make each derivative vanish separately, one after the other. This is the cld method of single parameter variation, where one seeks a minimum with respect to one variable at a time using one of the techniques described earlier. Of course when you have finished minimizing with respect to $x_2$, you may no longer be at a minimum with respect to $x_1$, so you generally have to start all over again, but the process usually does converge, as illustrated for two variables in this diagram. Here the curves represent
contours of equal function value, and the straight lines show the steps taken in minimizing $F$ with respect to $x_1$, then $x_2$, then $x_3$, etc. In this case the method converges nicely after only four single-parameter minimizations.

Consider now the function represented by the contours shown below. Here the method proceeds much more slowly because of the narrow valley.

Such behaviour in many dimensions causes this method to be generally considered as unacceptably slow.

Two of the more successful improvements aimed at avoiding such behaviour are due to Hooke and Jeeves$^3$) and Rosenbrock$^2$). We discuss the latter below.

3.3 Rosenbrock's method

Rosenbrock's algorithm$^2$) starts by performing single-parameter minimizations as above. Then when one full cycle of all parameters has been completed, a new set of orthogonal axes is defined with one axis taken as the vector from the start point to end point of the cycle. This vector points in the direction of previous over-all improvement and is expected to be a good direction for future improvement. In the case of the narrow valley seen above, it should point more or less along the valley and avoid the zig-zag behaviour. The next cycle of single-variable minimizations is performed using multiples of the newly defined axes as variables.
The Rosenbrock method generally performs well, being quite stable and capable of following narrow valleys, but as the number of variables increases, the efficiency drops, probably because the new axis defined by past improvement is based on points so far apart that it no longer points along a "hyper-valley" at the start point of the next cycle. Also, its terminal convergence is slow compared with the more "quadratic" methods described in Section 4.

Another technique, that of Davies, Swann, and Campey\textsuperscript{4}) (unpublished, see Ref. 4) is similar to Rosenbrock's and will not be described here.

3.4 The simplex method

One of the most successful stepping methods in many variables is that of Nelder and Mead\textsuperscript{5}), based on the simplex. A simplex is an \(n\)-dimensional figure specified by giving its \(n + 1\) vertices. It is a triangle in two dimensions, a tetrahedron in three, etc. The algorithm takes the name simplex because at each step the information it carries about the function consists of its values at \(n + 1\) points. One can easily visualize how the method works by considering the two-dimensional case as in the diagram below. The three starting simplex points are somehow chosen (perhaps randomly) and the function is evaluated at each point. Let the point \(P_H\) be that at which the function value is highest (worst) and \(P_L\) that at which it is lowest. Let \(\bar{P}\) be the centre-of-mass of all points in the simplex except \(P_H\); that is:

\[
\bar{P} = \frac{1}{n} \left\{ \sum_{i=1}^{n+1} P_i - P_H \right\}.
\]
From the original simplex, a new simplex is formed by replacing $P_H$ by a better point if possible. The first attempt to find a better point is made by reflecting $P_H$ with respect to $\overline{P}$, producing $P^* = \overline{P} + (\overline{P} - P_H)$. If $F(P^*) < F(P_L)$, a new point is tried at $P^{**} = \overline{P} + 2(\overline{P} - P_H)$. If $F(P^*) > F(P_H)$, a new point is tried at $P^{**} = \overline{P} - \frac{1}{2}(\overline{P} - P_H)$. The best of the new points then replaces $P_H$ in the simplex for the next step, unless none of them is better than $P_H$. In the latter case, a whole new simplex is formed around $P_L$, with dimensions reduced by a factor of 0.5.

Variations on the method are possible by using different contraction or expansion factors when searching along the line from $P_H$ through $\overline{P}$ (dotted in diagram). Another interesting possibility is to attempt a quadratic interpolation step along the dotted line whenever three points have been determined ($P_H$, $P^*$, $P^{**}$). However, one must be careful not to accept a point too close to $\overline{P}$, for then the simplex collapses into a line (or in general a hyperplane of $n-1$ dimensions) from which it can never recover.

The simplex algorithm, being designed always to take as big steps as possible, is rather insensitive to shallow local minima or fine structure in the function caused by rounding errors, statistical errors (Monte Carlo output), etc. Another of its virtues is that of requiring few function evaluations, usually one or two per iteration. In addition, each search is in an "intelligent" direction, pointing from the highest value to the average of the lowest values. Compare this with Rosenbrock's method, where really only the principal axis is an "intelligent" direction, and all other searches are for exploring along orthogonal axes to determine a new principal axis.

A convenient convergence criterion for the simplex method is based on the difference $F(P_H) - F(P_L)$. The iterations are stopped when this difference is less than a preset value. As a final step, the function is evaluated at $\overline{P}$, which is often slightly better than $F(P_L)$.

In view of the danger mentioned above -- of the simplex collapsing into a hyperplane of dimension $n-1$ -- it has been suggested to use $n+2$ or more points rather than $n+1$ at each step. I have tested this idea, which is equivalent to introducing a dummy parameter $c$ of which the function is independent, and have always found the efficiency of the algorithm to decrease under these conditions.
4. **Gradient Methods**

4.1 Calculating derivatives

I will call a *gradient method* any technique which uses information from a very small range of the variables (i.e. essentially derivatives) to predict good trial points relatively far away. This does not necessarily mean that they follow the gradient, but only that the gradient, and perhaps higher derivatives, are used or estimated.

It is of course possible in most cases to calculate analytically the numerical values of the derivatives of a function, just as it is possible to calculate the value of the function itself. However, it is often inconvenient and dangerous if the algebra is complicated, so that very often we are faced with minimizing a function for which no derivatives are provided. Since the most powerful algorithms discussed below require derivatives, a general minimization program must be able to estimate the derivatives of the function by finite differences.

A first derivative may be estimated from

\[
\frac{\partial F}{\partial x} \bigg|_{x_0} \approx \frac{F(x_0 + d) - F(x_0)}{d},
\]

where \(d\) is a "small" displacement. The error will be, to lowest order in the Taylor's expansion,

\[
\delta \approx \frac{d}{2} \cdot \frac{\partial^2 F}{\partial x^2} \bigg|_{x_0}.
\]

It is therefore advantageous to make \(d\) as small as possible, but still large enough so that the rounding error in the computation of \(F\) does not become larger than the error introduced by \(\delta\). Since the second derivatives may not be known, it may not be possible to find an optimum step-size \(d\), so we may just have to close our eyes and guess.

A much safer method would be to use points chosen symmetrically on either side of \(x_0\), giving

\[
\frac{\partial F}{\partial x} \bigg|_{x_0} \approx \frac{F(x_0 + d) - F(x_0 - d)}{2d},
\]
for in this case the error $\delta$ vanishes to second order and the lowest
order term is proportional to the third derivative. A disadvantage of
this method is that it requires 2n function calls to estimate the n first
derivatives, whereas the asymmetric steps require only n + 1 [or only n if
$F(x_0)$ has to be evaluated anyway]. An advantage of the symmetric steps
method, however, is that it gives the second derivatives as a by-product
[assuming $F(x_0)$ known]:

\[
\frac{\partial^2 F}{\partial x^2} \approx \frac{F(x_0 - d) + F(x_0 + d) - 2F(x_0)}{d^2},
\]

and from the relationship for the error $\delta$ in the asymmetric method, a
conservative upper limit of the uncertainty in the first derivative
results assuming at least that the symmetric formula gives a smaller
error than the asymmetric one. A complete treatment of step sizes is
beyond the scope of these lectures but can be found in a paper by
Stewart\textsuperscript{6}).

The numerical evaluation of second derivatives is facilitated by
the fact that they should be approximately constant over small regions,
so that symmetrical steps are usually not necessary. Unfortunately,
however, there are a lot of second derivatives to evaluate; since they
form a symmetric $n \times n$ matrix, there are $n(n + 1)/2$ independent compo-
nents, requiring at least $n(n - 1)/2$ points in addition to those
required for the symmetric deriv-
atives. For two parameters, a mini-
mum point pattern is shown in the
diagram at left. The odd point (for
the mixed second derivative) could
have been chosen in any corner. The
two-dimensional diagram is somewhat
misleading since for large n, the
number of "odd points" is n times
larger than the number of "symmetric"
points.
4.2 Steepest descent

As soon as the function's first derivatives are known, it is natural to follow the direction of the negative gradient vector in seeking a minimum, since this is the direction in which the function is decreasing the fastest. Such a technique was used by Cauchy more than a century ago, and is the basis of what is now known as the method of steepest descent.

This method consists of a series of one dimensional minimizations, each one along the direction of local steepest descent (gradient) at the point where each search begins. Of course the direction of the gradient is not constant along a line even for a general quadratic function, so we expect many iterations to be necessary, but the method can be shown to converge for a quadratic function. Let us follow its progress for a typical function whose contours are shown in the diagram. We immediately see an unfortunate property of the successive search directions: if each linear minimization is exact, successive searches must be in orthogonal directions. In two dimensions, this yields steps which look just like the single parameter variation method with the axes rotated to line up with the gradient at the start point. In many dimensions the situation is not quite so bad, but successive directions are still orthogonal and the algorithm cannot be considered acceptable.

It is in fact easy to draw contours for a reasonably well-behaved hypothetical function (as at left) where the direction to the minimum is just perpendicular to the gradient.
4.3 Newton's method

It is clear that since a general quadratic function is determined by specifying its value, first derivatives, and second derivatives at a point, it can be minimized in one step if and only if all this information (or its equivalent) is taken into account. Let us write a quadratic function as

$$F(x) = F(x_0) + \mathbf{g}^T \cdot (x - x_0) + \frac{1}{2} (x - x_0)^T \mathbf{G}(x - x_0),$$

where the gradient $\mathbf{g}$ is evaluated at $x_0$ and the second derivative matrix $\mathbf{G}$ is a constant. Then the minimum is given directly by

$$x_m = x_0 - \mathbf{G}^{-1} \mathbf{g} = x_0 - \mathbf{V} \mathbf{g},$$

where the inverse of the second derivative matrix is the covariance matrix $\mathbf{V}$.

This is then the many-dimensional equivalent of quadratic interpolation discussed earlier, and it is subject to the same sort of difficulties when applied as an iterative technique to general non-quadratic functions. But let us first point out its good features:

i) the step size is no longer arbitrary, but is prescribed precisely by the method;

ii) the step directions are no longer necessarily along the gradient vector but take account of parameter correlations (narrow valleys or ridges) through the mixed second derivative terms.

In practice, however, the method is unstable, essentially for the reasons given in Section 2.4. In particular, it diverges whenever the matrix $\mathbf{G}$ (or $\mathbf{V}$) is not positive-definite (see next section). In its unmodified form the method is used only when the minimum is known to be very close or when the function is known to be positive quadratic (for linear least squares). However, it is clearly a powerful technique and is worth studying in some detail since all the most successful algorithms are based on Newton-like steps, as discussed below.
4.4 Positive-definite quadratic forms

We pause here briefly to consider the properties of quadratic forms useful for understanding the more powerful gradient methods. In one dimension the description is simple; a general quadratic form can be written

\[ F(x) = a + gx + \frac{1}{2} Gx^2, \]

where \( g = \partial F/\partial x \) at \( x = 0 \), and \( G = \partial^2 F/\partial x^2 \) also at \( x = 0 \). This function has a minimum if and only if \( G \geq 0 \). If \( G = 0 \), the minimum is at infinity. The minimum (if it exists) is at \( x = -g/G \). When using a quadratic approximation to minimize a general non-linear function, it makes sense to take a step to \( x = -g/G \) only if \( G > 0 \) since otherwise we step to a predicted maximum or to infinity. A possible remedy if \( G < 0 \) is to take a step \( x = -g \); that is, to set \( G \) arbitrarily equal to unity so that the step will at least be in the right direction although it will now have arbitrary length. Consideration of the sketch below shows that this is the only thing we can do unless more information is available, since the quadratic part of the function is not convex or positive-definite at the point \( x_0 \):

![Sketch of quadratic function](image)

These arguments may now be extended to many dimensions where \( g \) becomes the gradient vector \( \overrightarrow{g} \), and \( G \) becomes the second derivative matrix \( \overrightarrow{G} \). Then the Newton step to \( \overrightarrow{x} = -\overrightarrow{G}^{-1}\overrightarrow{g} \) makes sense only if \( \overrightarrow{G} \) (or \( \overrightarrow{G}^{-1} \)) is a positive-definite matrix, since only then does the quadratic form
\[ F(x) = a + g^T x + \frac{1}{2} x^T G x \]

have a minimum. If \( G \) is singular, the predicted minimum (or maximum) is not unique.

Unfortunately there is no simple way of telling, in general, if a matrix is positive-definite by inspecting individual components, but we can at least state some of the many useful properties of such matrices. Two necessary (but not sufficient) conditions for a (square, symmetric) matrix to be positive-definite are:

i) the diagonal elements must be positive (this is in fact sufficient for a \( 1 \times 1 \) matrix);

ii) the off-diagonal elements must obey \( G^2_{ij} < G_{ii} G_{jj} \).

[Properties (i) and (ii) together are sufficient for a \( 2 \times 2 \) matrix.]

While the above conditions are easy to check, they are not in general sufficient. Some necessary and sufficient conditions are the following:

iii) All the eigenvalues of the matrix are positive. This is generally a rather difficult calculation and is usually approximate.

iv) The determinants of all the upper left square submatrices (formed as indicated in the diagram at left) are positive. This is probably the easiest method.

v) The scalar \( e^T G e \) is positive for all vectors \( e \). This is usually taken as the definition of a positive-definite matrix, and explains why a positive-definite matrix yields a quadratic form with a minimum: the function increases in all directions from \( e = 0 \).

vi) The inverse \( G^{-1} = \nabla \) is positive-definite.

Now suppose that \( G^{-1} \) is calculated for a Newton step and turns out to be non-positive-definite. In analogy to the one dimensional case we would simply take \( G = I \), the unit matrix, and the Newton step would become a steepest-descent step of arbitrary length, which is probably not so bad an idea and is in fact often done. But we can do better by
trying to make a positive-definite matrix which is as "close" as possible to the unacceptable $G$. The ways in which this can be done depend on what is "wrong" with $G$. For example:

i) If all the diagonal elements of $G$ are positive, the off-diagonal elements can simply be set $= 0$. The resulting matrix will be better than the unit matrix, since it will at least produce a scale-invariant step and non-arbitrary step length.

ii) If the only thing "wrong" with $G$ is that one or more off-diagonal elements of $G$ or $G^{-1}$ do not satisfy condition (ii) above, just these off-diagonal elements can be set to zero.

iii) The matrix $(G + \lambda I)^{-1}$ can be used instead of $G^{-1}$, where $\lambda$ is greater than the largest negative eigenvalue of $G$. This requires a large amount of calculation and so is not very convenient, but it is quite appealing since it amounts to taking a step which is intermediate between a Newton step and a steepest-descent step (for large values of $\lambda$ the step becomes short and in the direction of the gradient).

iv) If one or more of the diagonal second derivatives are negative, the non-positive-definiteness can be turned into an advantage, since it indicates a direction (or directions) in which the negative first derivative is increasing in magnitude rather than decreasing. This suggests an especially fruitful direction for a single-parameter-variation step which should not only lead to a good decrease of the function value but should also lead more quickly to a region of positive-definiteness.

Minimization methods based on variations of Newton's method as suggested by the above considerations are usually called quasi-Newton methods. Many such algorithms have been published and some are quite successful, but the field is still open for new ideas.

The principal drawback of such techniques is the repeated evaluation and inversion of the second-derivative matrix. The calculation of the second derivatives usually requires a rather long time, proportional to $n^2$, and the matrix inversion, although usually faster, increases with $n$ like $n^3$. 
One of the most interesting results concerning quadratic forms is the basis of a collection of related techniques described in the next sections, which do not require explicit repeated evaluations of $G$.

4.5 Conjugate directions

The vectors $d_i$ and $d_j$ are said to be conjugate with respect to a positive-definite symmetric matrix $A$ if

$$d_i^T A d_j = 0 \quad \text{for} \quad i \neq j.$$ 

If $A$ is the unit matrix $I$, the conjugate vectors $d$ would be orthogonal, so conjugacy can be thought of as a generalization of orthogonality. A set of $n$ conjugate vectors span an $n$-dimensional space, and any point in the space can therefore be expressed as a linear combination of $n$ conjugate vectors.

Although the matrix $A$ does not uniquely define a set of conjugate vectors, such a set can always be constructed by a procedure similar to the Gram-Schmidt orthogonalization method. Let us start for example with an arbitrary vector $d_1$. Then the vector

$$d_2 = A d_1 - \frac{d_1^T A d_1}{d_1^T A d_1} d_1$$

can be seen to be conjugate to $d_1$ since the product $d_1^T A d_2$ vanishes identically. The process can then be continued in the same way to construct a $d_3$ which will be conjugate to both $d_1$ and $d_2$, and so forth up to $d_n$.

Such vectors become interesting for minimization problems when they are conjugate with respect to the hessian (second derivative) matrix $G$. In this case a theorem of Fletcher and Reeves\(^7\) states that a sequence of linear minimizations in each of the $n$ conjugate directions will minimize a general quadratic function of $n$ variables. That this is true can be seen quite easily as follows. Let the quadratic function be

$$F(x) = F(0) + x^T G x + \frac{1}{2} x^T C x$$
and the n directions \( d_i \) be conjugate with respect to \( G \):

\[
d_i^T G d_j = 0, \quad i \neq j.
\]

Then the vectors \( x \) and \( g \) can be expressed as linear combinations

\[
x = \sum_i y_i d_i^\top,
\]

\[
g = \sum_i c_i d_i,
\]

so that the general quadratic becomes

\[
F(x) = F(Q) + \left( \sum_i c_i d_i^\top \right) \left( \sum_j y_j d_j \right) + \frac{1}{2} \left( \sum_i y_i d_i^\top \right) G \left( \sum_j y_j d_j \right).
\]

Now if the last term above is regrouped as a double sum, the terms with \( i \neq j \) drop out because of the conjugacy condition, so that the whole expression can be simplified as

\[
F(x) = F(Q) + \sum_i \sum_j c_i d_i^\top d_j y_j + \frac{1}{2} \sum_j y_j^2 d_j^\top G d_j
\]

\[
= F(Q) + \sum_j \left( b_j y_j + b_j' y_j^2 \right)
\]

where

\[
b_j = \sum_i c_i d_i^\top d_j
\]

and

\[
b_j' = d_j^\top G d_j
\]

are constants. By expressing the quadratic in terms of \( y \) instead of \( x \), we have separated it into a sum of independent one-parameter quadratic functions. A minimization with respect to \( y_i \) (a linear minimization along the direction \( d_i \)) will therefore be independent of the minimizations along the other conjugate directions, which demonstrates the validity of the theorem.
The above theorem tells us what is "wrong" with the single-parameter-variation method: we should be using conjugate directions rather than simply orthogonal axes. However, since the construction of conjugate vectors seems to require knowledge of the hessian \( \mathcal{G} \), this does not yet help very much in practice, for if we knew \( \mathcal{G} \) (and \( \mathbf{g} \)) we could minimize a quadratic immediately by means of Newton's method, and would not need to use \( n \) linear minimizations.

The usefulness of conjugate directions comes from the fact that there are ways of determining such directions implicitly, without first evaluating the entire hessian matrix \( \mathcal{G} \). Of course, by the time all \( n \) conjugate directions are determined, by whatever method, information equivalent to the matrix \( \mathcal{G} \) must have been determined. However, by that time considerable minimization may already have been performed, as in the method implied by the following theorem.

If \( \mathbf{x}_0 \) and \( \mathbf{x}_1 \) are minimum points in two parallel subspaces, then the direction \( \mathbf{x}_1 - \mathbf{x}_0 \) is conjugate to any vector which lies in either subspace. This can easily be seen in two dimensions as illustrated in the figure at left. Since \( \mathbf{x}_0 \) is a minimum along the direction \( \mathbf{d}_1 \), the gradient of \( F \) at \( \mathbf{x}_0 \) must be orthogonal to \( \mathbf{d}_1 \):

\[
\mathbf{d}_1^T(\mathbf{g} + \mathcal{G}_{\mathbf{x}_0}) = 0,
\]

where \( \mathbf{g} \) is the gradient at \( \mathbf{x} = 0 \). Similarly at \( \mathbf{x}_1 \):

\[
\mathbf{d}_1^T(\mathbf{g} + \mathcal{G}_{\mathbf{x}_1}) = 0.
\]

Subtracting the above equations, the first terms drop out and we have:

\[
\mathbf{d}_1^T\mathcal{G}(\mathbf{x}_1 - \mathbf{x}_0) = 0,
\]

showing that \( (\mathbf{x}_1 - \mathbf{x}_0) \) is conjugate to \( \mathbf{d}_1 \).

Unfortunately, extending this algorithm to three dimensions requires three additional minimizations in order that the third direction be conjugate to both of the first two, so that convergence for a general
quadratic in n variables is obtained only after n iterations involving in all n(n+1)/2 linear minimizations. Since this is just the number of independent elements in the second derivative matrix, we would be better off for quadratic functions to calculate this matrix directly and avoid the linear searches. On the other hand, for non-quadratic functions the conjugate directions method should be much more stable since it proceeds by a series of linear searches in independent directions and still guarantees convergence in a finite number of steps once a quadratic region is entered. In addition, this method has the advantage of requiring neither first nor second derivatives of the function. (Strictly speaking, then, it should have been discussed in Section 3 rather than in this section.)

A disadvantage of the algorithm described above is that for each iteration, n minimizations are performed in direction $d_1$, whilst only one is performed in direction $d_n$. This undesirable asymmetry is largely avoided in a variation due to Powell$^8$).

4.6 Conjugate gradients

When the first derivatives of the function are calculated, a somewhat more elegant method can be used, known as the method of conjugate gradients$^7$). Suppose that the function and its gradient are evaluated at two points $x_0$ and $x_1$, giving differences:

$$\Delta x = x_1 - x_0$$
$$\Delta g = g_1 - g_0 .$$

Then if the function were quadratic with hessian $G$ we would have

$$\Delta g = G \Delta x .$$

Any vector $d_1$ orthogonal to $\Delta g$ would then be conjugate to $\Delta x$: 

$$d_1^T \Delta g = d_1^T G \Delta x = 0 ,$$

which immediately suggests a method for obtaining conjugate directions without knowing $G$, based on the change in gradient along a previous direction.
In the method of conjugate gradients, successive one-dimensional minimizations are performed along conjugate directions with each direction being used only once per iteration. The first direction is taken as \( d_0 = -g_0 \), the steepest descent vector at \( x_0 \). Let the minimum along this direction be at \( x_1 \) where the gradient is \( g_1 \). Then the next search direction \( d_1 \), which we want to be conjugate to \( d_0 \), must be a linear combination of the only vectors we have at hand, namely:

\[
d_1 = -g_1 + b d_0 .
\]

The conjugacy condition is

\[
d_1^T g_0 = d_1^T (x_1 - x_0) = 0
\]

or

\[
(-g_1^T + b d_0^T) g_0 = (-g_1^T - b g_2^T) (g_1 - g_0) = 0 .
\]

Since \( x_1 \) is a minimum along direction \( d_0 = -g_0 \), the direction \( g_0 \) is orthogonal to the gradient at \( x_1 \), so that \( g_1 g_0 = 0 \). We are then left with

\[
b = \frac{g_1^T g_1}{g_0^T g_0}
\]

so that the new conjugate direction is

\[
d_1 = -g_1 + \left( \frac{g_1^T g_1}{g_0^T g_0} \right) d_0 .
\]

This process can be continued to generate \( n \) directions, each one conjugate to all the others. It turns out that the same simple formula holds for all the successive conjugate directions

\[
d_{i+1} = -g_{i+1} + \left( \frac{g_{i+1}^T g_{i+1}}{g_i^T g_i} \right) d_i .
\]
4.7 Variable metric methods (VMM)

In analogy with the methods of differential geometry and general relativity, it is convenient to consider the properties of the function $F(x)$ as being in fact properties of the space of the variables $x$. We have already made some rudimentary use of this idea when we generalized from the usual orthogonal coordinate axes to a system defined by axes pointing in conjugate directions. We now wish to go further and be able to express the properties of the function $F$ geometrically as the properties of the non-Euclidean space of its variables $x$.

The fundamental invariant in a non-Euclidean space is the squared distance element

$$ds^2 = dx^T A dx,$$

where $dx$ is a differential coordinate displacement and $A$ is the covariant metric tensor which determines all the properties of the space under consideration. When $A$ is just the unit matrix $I$, the above formula for $ds^2$ just expresses the Pythagorean theorem for an $n$-dimensional Euclidean space. When off-diagonal elements of $A$ are non-zero and when the elements are allowed to vary as functions of $x$, a generalized non-Euclidean space is generated.

It is easily verified that the second derivative (hessian) matrix $G$ behaves under coordinate transformations like a covariant tensor and we will identify it with the metric tensor of our space. The inverse $V = G^{-1}$ is a contravariant tensor and becomes the contravariant metric tensor. (For a discussion of covariant and contravariant tensors, see for example chapter 10 of reference 9.) This immediately enables us to construct two scalar (invariant under coordinate transformations) quantities:

a) $ds^2 = dx^T G dx$

is the square of the generalized distance between the point $x$ and the point $x + dx$. When $F$ is a chi-square function which is minimized to determine some best parameters $x$, then the physical meaning of the generalized distance $ds$ is just the number of "standard deviations" $x + dx$ is away from $x$. That is, the use of the metric tensor $G$ enables us to scale the distance $dx$ so that it comes out as a physically (or
statistically) meaningful invariant quantity instead of being expressed in arbitrary units (or a mixture of arbitrary units!).

And

$$\rho = g^T V g$$

is twice the difference between the function value at the point where $V$ and the gradient $g$ are calculated and the minimum of a quadratic form with hessian matrix $G = V^{-1}$. That is, $\rho/2$ is the expected (vertical) distance to the minimum if the function $F$ were quadratic. This provides us with an important scale-free convergence criterion for any method which provides approximations to $V$ and $g$.

When the function $F$ is quadratic, $G$ is constant everywhere and, in the sense outlined above, this is equivalent to working in a space with a constant metric. For real non-linear functions we expect higher-order terms to be small but not negligible, so that we can think of working in a space with a slowly-varying metric tensor. Minimization methods based on this approach are known as variable metric methods. They differ from the basic Newton-Raphson method in that the matrix $G$ is not completely re-evaluated at each iteration, but is assumed to be well approximated by taking the $G$ of the previous iteration and applying a correction based on new information from the current iteration. This correction is known as the matrix updating formula, which in general differs from method to method.

Variable metric methods therefore proceed generally by the following steps:

i) A starting point $x_0$ is given, the gradient $g_0$ at that point is calculated, and some approximation to $G^{-1}$, say $V_0$, is constructed. The starting $V_0$ may be only the unit matrix, or it may actually be the inverse of the full second derivative matrix.

ii) A step is taken to $x_1 = x_0 - V_0 g_0$, which would be the minimum if $F$ were quadratic and if $V_0$ were the true covariance matrix. Since $x_1$ is not the position of the minimum in the general case, it is usual to perform a linear search along this direction, finding the $\alpha$ which minimizes $F(x_0 - \alpha V_0 g_0)$. In either case let the new point be called $x_1$ and let the gradient calculated at $x_1$ be $g_1$. 
iii) The matrix $V$ is corrected using an updating formula of the form

$$V_1 = V_0 + f(V_0, x_0, x_1, g_0, g_1).$$

Then $g_0$ is replaced by $g_1$, $x_0$ by $x_1$, and $V_0$ by $V_1$, and steps (ii) and (iii) are repeated until some convergence criteria are satisfied.

The different methods differ chiefly in the choice of updating function $f$, as described in the following sections, and in the extent to which linear minimizations are necessary. Less important variations involve the starting approximation $V_0$ and various safeguards against "unreasonable" steps and non-positive-definiteness as for the Newton techniques.

4.8 Davidon's rank-two formula

Probably the first -- and perhaps still the best -- variable metric method was developed in 1959 by Davidon and later published in simplified form in 1963 by Fletcher and Powell\(^{10}\). Davidon's updating formula for the covariance matrix is the following:

$$V_1 = V_0 + \frac{\delta \delta^T}{\delta^T \gamma} - \frac{V_0 \gamma \gamma^T V_0}{\gamma^T V_0 \gamma},$$

where the changes in position and gradient on the last step were

$$\delta = x_1 - x_0$$

and

$$\gamma = g_1 - g_0,$$

and $V_0$ was the previous estimate of the covariance matrix. This is called a rank-two formula since the correction $V_1 - V_0$ is a matrix of rank two in the space of $\delta$ and $V_0 \gamma$, as can be seen directly by inspection of the formula.

One fundamental requirement of an updating formula is that the new matrix satisfies the relationship

$$V_1 \gamma = \delta,$$

since $\gamma = G \delta$ for a quadratic with hessian $G$. It is easily seen that Davidon's formula satisfies this requirement:
\[ Y_1 Y = \left[ Y_0 + \frac{\delta \delta^T}{\delta^T Y} - \frac{Y_0 Y Y^T Y_0}{Y^T Y Y^T Y_0} \right] Y \]

\[ = Y_0 Y + \frac{\delta \delta^T Y}{\delta^T Y} - \frac{Y_0 Y Y^T Y_0 Y}{Y^T Y Y^T Y_0} \]

\[ = Y_0 Y + \delta - Y_0 Y = \delta . \]

An unfortunate feature of the Davidon algorithm is the need to perform at each iteration a linear minimization along the direction given by a Newton step, \(-V_0\). This linear search step is, however, necessary in order to assure convergence for general functions. Fletcher and Powell show\(^{10}\) that if the starting approximation to \(Y\) is positive-definite, then \(Y\) will remain positive-definite after all updatings, but they have to use the fact that each iteration is a linear minimization, that is

\[ \frac{T}{\delta i V_0 \delta_0} = 0 . \]

It can be shown that this method is quadratically convergent, at most \(n\) iterations (\(n\) linear searches and \(n\) gradient calculations) being required for an \(n\)-dimensional quadratic form.

4.9 The rank-one formula

In an effort to avoid the linear minimizations required by Davidon's algorithm, several workers have independently developed an interesting updating formula of rank one. In this case Davidon in 1968 was the first to publish an algorithm\(^{11}\) based on the formula, and Powell\(^ {12}\) has summarized the properties of this formula and of algorithms based on it.

The rank-one updating is:

\[ Y_1 = Y_0 + \frac{(\delta - Y_0 Y)(\delta - Y_0 Y)^T}{Y^T (\delta - Y_0 Y)} . \]

It can be shown\(^ {12}\) that this is the only formula of rank two (or less) for which not only \(Y_1 Y = \delta\) but:

\[ Y_1 Y_i = \hat{\delta}_i , \]
where \( \delta_1 \) and \( \gamma_1 \) are the step and gradient changes at any previous iteration. This is known as the hereditary property, since \( \gamma_1 \) can be said to inherit the fundamental property \( V\gamma = \delta \) with respect to all previous iterations (up to \( n \)).

The hereditary property assures that after \( n \) iterations, \( \gamma_1 \) will be the true covariance matrix if \( F \) is quadratic, no matter what steps were taken (almost), so that if Newton steps are taken, convergence for a quadratic function is assured after \( n \) iterations, without the need for linear minimizations.

In addition, the rank-one formula is symmetric, in the sense that the expression for \( V\gamma_1 \) in terms of \( V_0 \) is the same as that for \( V_1 \) in terms of \( V_0 \), provided \( \delta \) and \( \gamma \) are interchanged. The meaning of this symmetry property will be discussed in the next section.

But, as nothing is perfect, so the elegance and mathematical beauty of the rank-one formula hide a number of numerical and practical difficulties which can make it highly unstable when applied to a general function. In particular, if the vector \( \gamma \) happens to be orthogonal to the vector \( (\delta - V_0 \gamma) \), the denominator goes to zero in the updating formula, and an unbounded correction is possible. Since these vectors may be orthogonal, even for a quadratic function, the problem of numerical instability is a serious one.

Moreover, the matrices \( V_1 \) do not really converge to the true covariance matrix in the usual meaning of the term convergence. Although it is true that \( V_1 \) will be equal to the true covariance matrix at the \( n \)th step for a quadratic function (barring numerical difficulties), the intermediate matrices \( V \) may vary wildly from step to step, so that on any particular iteration \( V_1 \) may be a rather poor approximation. This is especially dangerous when the function is not quadratic, since the large corrections necessary in later iterations will generally not compensate properly the fluctuations in early steps. Also, there is no guarantee that intermediate matrices will remain positive-definite, and hence no guarantee of a reduction in the value of \( F \) at each step, even for a quadratic \( F \).

All these difficulties can, of course, be overcome by programming enough safeguards into the algorithm, but this can only be done at the
expense of efficiency and sometimes only by abandoning temporarily the updating formula itself, which makes it lose some of its appeal. Different approaches are possible depending on whether it is considered important to maintain positive definiteness as in the Davidon algorithm\textsuperscript{11}, or important not to abandon the exact rank-one formula as in Powell's method\textsuperscript{12}).

4.10 Fletcher's unified approach to VMM

The existence of two different updating formulas with very different properties generated a lot of interest in variable metric methods (VMM) during the years 1967-1971, since it showed VMM to be very promising and left many questions unanswered, such as:

i) How can it be that the rank-one and rank-two formulas have such different properties? What is the relationship between them?

ii) Is there a way to combine the best properties of both formulas?

iii) Are there other good formulas? Is it possible to define a class of "admissible" formulas?

A certain understanding of the above problems has recently been made possible by the work of a number of people. In particular, a recent paper by Fletcher\textsuperscript{13}) presents a unified approach to VMM, which will be given here.

Recall that the rank-one equation is symmetrical (in a sense defined in Section 4.9), but as we shall now see, the rank-two formula is not. Indeed the asymmetry suggests a way to construct a possible third formula by taking the "mirror image" of the rank-two formula. The basic idea is that a new formula should satisfy the fundamental relationship

$$\mathbf{V}_1 \mathbf{Y} = \delta,$$

and therefore its inverse should satisfy

$$\mathbf{Y} = \mathbf{V}_1^{-1} \delta.$$

We can indeed write down the updating formula for $\mathbf{V}_1^{-1}$ which corresponds to the rank-two formula for $\mathbf{V}_1$:

$$\mathbf{V}_1^{-1} = \left( I - \frac{\gamma_Y^T}{\delta^T Y} \right) \mathbf{V}_1 \left( I - \frac{\delta^T Y}{\delta^T Y} \right) - \frac{\gamma_Y^T}{\delta^T Y}.$$
This matrix $V_1$ can now be thought of as a mapping from $\delta \rightarrow \gamma$ since $\gamma = V_1 \delta$. If we interchange $\gamma$ and $\delta$ in the formula, it will then give a mapping from $\gamma \rightarrow \delta$, thereby producing a new updating formula where $V_1 \gamma = \delta$. The new dual formula will be just

$$V_1 = \left( I - \frac{\delta T}{\delta T \gamma} \right) Y_0 \left( I - \frac{\gamma T}{\gamma T \gamma} \right) + \frac{\delta T}{\delta T \gamma}.$$ 

If we try this trick with the rank-one formula, we just get the same rank-one formula back again, since it is symmetric in this sense, or dual to itself. But with the rank-two formula, the process of inverting and interchanging yields a new formula, also of rank-two, which is also a valid updating formula in the sense that it gives rise to a quadratically convergent WMM algorithm.

Now we go further and consider the class of formulas which includes both rank-two and dual formulas as special cases. Let us introduce the notation

$$V_1 = T(Y_0) \quad \text{for the rank-two formula},$$

and

$$V_1 = D(Y_0) \quad \text{for the dual formula},$$

and consider the class of updating expressions as introduced by Fletcher\textsuperscript{13}:

$$V_\phi = (1 - \phi)T + \phi(D),$$

where $\phi$ is some parameter which determines the exact formula. [Broyden\textsuperscript{14}, using a somewhat different notation, has also considered the same class of formulas.]

It then turns out that the rank-one formula is also in this class, with

$$\phi(\text{rank-one}) = \frac{\delta T \gamma}{\delta T \gamma - \gamma T Y_0 \gamma}.$$ 

Having now constructed a wide class of updating formulas, which in fact includes all formulas known to the author, it will prove interesting to consider their properties as a function of the generating parameter $\phi$. Probably the most important property, and the only one we will consider here, is that of monotonic convergence of $V$ toward the true covariance.
matrix for a quadratic function. [This is called Property 1 in Fletcher's paper\textsuperscript{13}, which should be consulted for details of the definition and for theorems concerning it.] The use of an updating formula with this property will guarantee an improvement in the approximation $V$ at each iteration (for a quadratic function).

Any formula $V_\phi$ with $\phi$ in the interval $[0, 1]$ possesses the monotonic convergence property. Such a formula is said to belong to the convex class of formulas. For any $V_\phi$ with $\phi$ outside the range $[0, 1]$, there exists some quadratic function for which $V$ diverges from the true covariance matrix.

From what we have already seen about the rank-one formula, it is not surprising to find that it does not belong to the convex class. Since $\nabla^T V > 0$ for any step which is an improvement, and since $\nabla^T V_0 V > 0$ if $V_0$ is positive-definite, it can be seen immediately from inspection of the equation for $\phi$(rank-one) that it must either be less than zero or greater than one.

The above considerations lead Fletcher to propose a new algorithm\textsuperscript{13} which is probably the most elegant and powerful of any VMM algorithm. Basically, he uses the general updating formula $V_\phi$, with the value of $\phi$ chosen according to the following scheme: If $\phi$(rank-one) $< 0$, set $\phi = 0$, corresponding to the usual rank-two formula. If $\phi$(rank-one) $> 1$, set $\phi = 1$, corresponding to the dual formula. In this way, one always uses a formula in the convex class, and chooses that one which is "closest" to the rank-one formula. It seems that the linear searches can then be eliminated and replaced simply by Newton's steps, unless the function is highly non-quadratic. The latter condition can easily be detected by comparing the actual improvement with the expected improvement at each iteration.

5. **SPECIALIZED TECHNIQUES**

All the methods outlined so far in these lectures are of rather general applicability, the only assumption being -- for some methods -- a predominantly quadratic behaviour in the immediate vicinity of the minimum. In order to develop more powerful methods than those already presented, we will have to give up some of this generality and exploit particular features of the functions to be minimized. In this section
we discuss a few specialized techniques which are still of rather wide applicability in the sense that most functions of physical interest fall in one or more of these classes.

5.1 Chi-square minimization

Probably the most common application of minimization in scientific research is in least squares fitting, where the function to be minimized is the sum of squares of deviations, between measured values and predictions of a model containing variable parameters:

$$F(\mathbf{x}) = \sum_{k=1}^{K} f_k^2(\mathbf{x}) = \sum_{k=1}^{K} \left( \frac{Y_k - T_k(\mathbf{x})}{\sigma_k} \right)^2,$$

where $Y_k$ and $\sigma_k$ are measured values and errors, and $T_k(\mathbf{x})$ are the values predicted by the model, depending on some parameters $\mathbf{x}$. Minimizing $F$ then yields best values (estimates) of the $n$ parameters $\mathbf{x}$, based on $K$ measurements $Y$ with random errors $\sigma$, where $K$ must be greater than or equal to $n$, and is usually much greater than $n$.

Let us now consider the second derivative matrix for $F(\mathbf{x})$, expressed in terms of the individual $f_k(\mathbf{x})$:

$$\frac{\partial^2 F}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i} \left( \frac{3}{\partial x_j} \sum_k f_k^2 \right)$$

$$= \frac{\partial}{\partial x_i} \left( \sum_k 2f_k \frac{\partial f_k}{\partial x_j} \right)$$

$$= \sum_k 2 \frac{\partial f_k}{\partial x_i} \frac{\partial f_k}{\partial x_j} + \sum_k 2f_k \frac{\partial^2 f_k}{\partial x_i \partial x_j}.$$

In the above r.h.s., it is usual to make the approximation that the second sum, involving second derivatives, is small compared with the first term involving products of first derivatives. This is called linearization. [Note that it is the model $T(\mathbf{x})$ that is being linearized, not the function $F(\mathbf{x})$.] In the important special case of linear least squares, the second sum is exactly zero, so that $F(\mathbf{x})$ is quadratic, and the whole minimization problem reduces to the inversion of the above matrix $\partial^2 F/\partial x_i \partial x_j$ (i.e. the taking of one Newton step).
In the more general case of non-linear least squares, the linearization approximation consists in taking

\[
\frac{\partial^2 F}{\partial x_i \partial x_j} = \sum_k 2 \frac{\partial f_k}{\partial x_i} \frac{\partial f_k}{\partial x_j}.
\]

This has the advantage of being easy to calculate and, moreover, it is always positive-definite (under rather weak conditions such as the existence of the derivatives, and provided it is non-singular). In fact in many cases the use of the above approximation in computing Newton steps is actually more effective than using the exact second derivative matrix because of the positive definiteness. Of course it must be remembered that the covariance matrix obtained by inverting this approximate matrix does not in general converge to the true covariance matrix even though the minimization based on it may converge to the true minimum.

5.2 Likelihood maximization

An increasingly important alternative to the least squares method in data fitting is the method of maximum likelihood. In this case the function to be minimized is of the form

\[
F(x) = - \sum_{k=1}^K \ln f_k(x),
\]

that is, a sum of logarithms. Here again, an approximation for the second derivative matrix can be found which involves only products of first derivatives:

\[
\frac{\partial^2 F}{\partial x_i \partial x_j} = - \frac{\partial}{\partial x_i} \frac{\partial}{\partial x_j} \sum_k \ln f_k \\
= - \frac{\partial}{\partial x_i} \sum_k \frac{1}{f_k} \frac{\partial f_k}{\partial x_j} \\
= - \sum_k \frac{1}{f_k^2} \frac{\partial f_k}{\partial x_i} \frac{\partial f_k}{\partial x_j} - \sum_k \frac{1}{f_k} \frac{\partial^2 f_k}{\partial x_i \partial x_j}.
\]
As with least squares, we can neglect the second sum, involving second derivatives. In the case of the likelihood function, the second derivatives of $f$ are never exactly zero over any finite range (exactly linear maximum likelihood does not exist, essentially because the likelihood function must be normalized so that its integral over the space of measurements is independent of the parameters $\mathbf{x}$). However, the approximation

$$\frac{\partial^2 F}{\partial x_i \partial x_j} \approx \sum_k \frac{1}{k^2} \frac{\partial f_k}{\partial x_i} \frac{\partial f_k}{\partial x_j}$$

has the same advantages as in the non-linear least squares case, namely speed of calculation and assured positive-definiteness.

5.3 Models with separable computing

It often happens that the computation of the function value $F$ can be arranged so that large parts of the calculation depend on only a few of the variable parameters $\mathbf{x}$. These parts will then remain unchanged if the corresponding parameters have not changed since the previous function evaluation. An important special case of this is when the calculation can be separated into $n$ pieces, each depending on only one parameter.

Whenever the computing is separable in the above sense, large portions of the computation may be avoided by testing which parameters have not varied since the previous function call, and using the previous results of the corresponding sub-calculation when appropriate. The overall saving in computer time will then depend, of course, on the minimization method used, and some otherwise inferior methods, such as single-parameter variation, may become relatively efficient because of the time saved in computing the function.

In particular, the cost of computing derivatives by finite differences will generally be much lower when the computing is separable, and in the extreme case of complete separability, all $n$ first derivatives may be computed in a time comparable with that of one full function evaluation.
5.4 Sets of related problems

Many applications involve a series of minimizations for which the
functions involved are closely related. For example, in determining
confidence intervals for a parameter \( y \) in a statistical problem involving
additional parameters \( x \) also to be estimated, one determines the curve
\( p(y) \) traced by the minima of the chisquare function \( F \) with respect to \( x \)
for different values of \( y \):

\[
p(y) = \min_{x} F(x,y).
\]

A series of points on this curve is then determined by fixing \( y \) at
several different values, and for each of these, minimizing \( F \) as a func-
tion of \( x \). Clearly then, information from the first minimization can be
used as a starting point for the second, and then extrapolated to get the
starting point for the third, and so on. In particular, one does not
expect the covariance matrix to vary considerably from point to point,
so this is an especially valuable piece of information to carry over from
one problem to the next.

Similarly, if an experiment is repeated with new, independent, but
identically distributed data, the minimizations involved in the data
analysis can be speeded up by supplying covariance matrices from the
analysis of the first experiment.

6. LOCAL AND GLOBAL MINIMA

6.1 The problem of multiple minima

All the methods presented so far have been designed to find a local
minimum, without any consideration of whether or not other local minima
exist, or whether the minimum found is actually the global minimum.
If the function has more than one local minimum, there is not even any
guarantee that these methods will find the minimum closest to the
starting point, let alone the global minimum. In fact, it is usually
assumed, when using these algorithms, that the function is unimodal
(has one minimum) in the region of interest likely to be explored during
the minimization.
Whenever the function may have more than one local minimum, new problems arise in addition to the problem of local minimization. First of all, the user must decide what he wants to know about the function. The following four possibilities are the most common and will be discussed here:

i) it is sufficient to know the location of any one local minimum;  
ii) only the global minimum is of interest;  
iii) only one minimum is of interest (the "physical solution"), but it need not be the global minimum; or  
iv) all local minima, including the global one, must be found and catalogued.

The first possibility, (i), is quite rare, but is easy to deal with, since any local minimization routine is sufficient.

Possibility (ii) is much more common, particularly in system optimization where the cost must be the smallest possible, not just small compared with other near-by solutions. Several methods exist for finding global minima, of which two will be discussed in the next sections. All such methods suffer from the absence of a stopping rule: even if the global minimum is found there is no way of recognizing it unless the function is known to be bounded and has reached its lower bound.

Possibility (iii) often arises in scientific research where the approximate values of some parameters are known in advance and one seeks a solution not too far from these values, corresponding to "the right valley" where the function may have several faraway valleys which may be deeper. The usual technique for making sure of staying in the right valley is first to fix the approximately known parameters at their assumed values and minimize with respect to all other variables, then starting from this point minimize in the entire variable space.

Possibility (iv), of having to find and record all local minima, is the most difficult of all. It arises, for example, in energy-dependent phase-shift analyses where all "solutions" are recorded at each energy, and a continuous set of solutions is sought, one at each energy, which have a smooth energy dependence. Although the techniques described below
may help in this problem, no exhaustive method is known to the author except for the prohibitive one of using many starting points equally spaced on an n-dimensional grid.

6.2 The Gelfand algorithm

Relatively few minimization methods are specifically designed for non-local search in many parameters. Probably the most successful of the ad hoc stepping methods is that of Gelfand\textsuperscript{15}). It is non-local because it provides a natural way to allow for function increases as well as decreases in any one step, while tending generally to decrease the function value.

The procedure is as follows. From the starting point $x_0$, a local minimization is begun (for example along the gradient) until the function differences between steps become small (at the point $a_0$). Then, going back to the starting point, a "long" random step is taken to the point $x_1$, and another rough local minimization is performed to reach the point $a_1$ (see figure above). Then the so-called "precipitous step" is taken along a line from $a_0$ to $a_1$, some distance past $a_1$ to $x_2$. Then from $x_2$ another rough local minimization is performed, yielding $a_2$, and another precipitous step is taken from $a_1$ past $a_2$ to $x_3$, and the search continues in this way.

The choice of the "precipitous step" length is important in determining whether the method will "roll over small ridges, but skirt a high mountain", as its authors say it should. But no precise way is given, except that "the choice of the length of the precipitous step is carried out experimentally (by trials) and it constitutes an important characteristic of the function".
Moreover, there is no stopping rule, since the method is essentially searching rather than converging. In practice one usually stops after a given length of computer time, but one would also stop if the program went around in circles repeating itself (which is very possible but not so easy to detect) or if a predetermined "acceptably small" function value was attained. This problem of stopping seems to be common to all non-local minimization methods.

6.3 The Goldstein-Price method

Goldstein and Price\textsuperscript{16} have proposed an elegant yet simple method for seeking other local minima after one local minimum has been found. It is based on a consideration of the analytic (Taylor series) properties of the function. Let us assume that the function can be represented as a Taylor series about a local minimum \( \bar{x}_1 \), where the first derivatives vanish:

\[
F(x) = F(\bar{x}_1) + \frac{1}{2} (x - \bar{x}_1)^T \mathcal{G}(x - \bar{x}_1) + \text{h.t.}.
\]

Now the higher terms (h.t.), involving third and higher derivatives, are important since these are the terms that will give rise to other local minima. In fact, we seek a way of transforming the function so that only the higher terms remain. Such a transformed function is \( F_1 \) such that:

\[
F_1(x_1, \bar{x}) = \frac{2(F(x) - F(\bar{x}_1))}{(x - \bar{x}_1)^T \mathcal{G}(x - \bar{x}_1)} = 1 + \text{h.t.}.
\]

By means of this transformation, we have "removed" the minimum at \( \bar{x}_1 \), and the way is cleared to search for other minima generated by the higher terms of the expansion about \( \bar{x}_1 \). The method therefore consists of seeking a local minimum of the function \( F_1 \). (It is required to know the second derivative matrix \( \mathcal{G} \) at the local minimum \( \bar{x}_1 \).) Since the quadratic form \( (x - \bar{x}_1)^T \mathcal{G}(x - \bar{x}_1) \) is always positive for positive-definite \( \mathcal{G} \), the function \( F_1 \) will become negative as soon as an improvement on \( \bar{x}_1 \) is found. Then starting from this improved point, the original function \( F \) can be minimized locally to yield a new, improved local minimum of \( F \).

If the minimum value found for \( F_1 \) is positive, then it may correspond to a new local minimum of \( F \), but not an improvement over \( \bar{x}_1 \).
In this case the procedure may be continued from this new point, forming a new function \( F_2 \), related to \( F_1 \) just as \( F_1 \) was related to \( F \). As usual, no stopping rule is given by the theory.

The method seems to work in practice, although experience with it is limited and no conditions are known under which it is guaranteed to work. It is appealing for reasons of its elegance and simplicity, and could prove to be an important tool in global minimization.

7. **CONCLUSION**

7.1 **Global strategy**

After having studied the properties of many minimization methods, we are finally faced with the problem of choosing one of them. As we have already stated, no one method can be optimum in the sense of being best for all functions. And even for one given function, it is unlikely to find a method which works well in all regions, far from the minimum as well as near.

All this suggests that we should try to tailor the programs to our function's needs. If we expect to be doing a lot of minimization, several programs should be prepared, based on different methods with properties suited to different kinds of functions. Then a decision as to which method to use would depend on a consideration of the particular function to be minimized. A decision tree, enabling the user to choose a method for his function, has been given in the review of Fletcher\(^{17}\). Certainly no two experts would agree on the details of such a logic diagram, since everyone has his own personal preferences in this field, but the general idea is a good one and indicates a way in which a routine could be chosen.

In large problems, the properties of the function may change drastically from one region to another, so that a good method far from the minimum, for example, may converge very slowly in the vicinity of the minimum. In particular, the simplex and Rosenbrock methods are quite insensitive to the exact shape of the function, and so should work as well in non-quadratic regions as in quadratic regions. However, once the region of the minimum is reached, the function should
be reasonably quadratic so that a method with quadratic convergence should certainly be used.

With the present state of the art, the decision as to which method to use, or when to change methods if more than one is used, must be based on a priori knowledge of the function. Ultimately one could imagine designing a super-algorithm which could examine the function or follow the progress of the minimization and choose the best method on the basis of what it finds out, rejecting those whose progress is too slow in favour of more suitable techniques. At present this is sometimes done, but only in a most rudimentary way, in the sense that some algorithms can tell when they are going astray and can signal this to a main program which can then try a safer method.

7.2 Computer programs

The purpose of this last section is not to suggest particular programs which may be available for general use, but rather to indicate in very general terms several ways in which such programs may be organized. The implementation and use of the techniques described in these lectures implies their being programmed for high-speed computers, with the program structure depending again on the nature of the problem to be solved.

One traditional program structure is that of the large autonomous program which as far as possible takes care of all details of initialization, input-output formats, error returns, and other organizational logic. The function to be minimized is submitted by the user in the form of a subroutine which returns a function value $F$ on each call, depending on the values of the formal parameters $x$. Another formal parameter is a flag informing the function subroutine when the first and last calls are being made, so that the subroutine may do any private initialization or printing of final results if necessary. The starting values and step sizes of the function parameters are usually read in by the main program on data cards, and there may be more data cards specifying which of several options are requested if the main program is capable of performing a variety of different tasks. Programs organized in this way are popular in laboratories where the principal task to be performed in a typical
job is a single large minimization problem, for then the main program can easily contain logic permitting change-over from one algorithm to another in case of failure to converge, and can contain many other features which relieve the user of a large amount of trivial "housekeeping" work.

A different approach is that of the small minimizing subroutine, designed to save memory space and to allow the user maximum flexibility. In this case the user must write the main program which calls the minimizer, as well as the function subroutine called by the minimizer. In return for the extra work, he then has complete control over the organizational details, and the minimizer is more likely to be machine-independent when available in a higher-level language such as FORTRAN. This approach is especially adapted to jobs consisting of a series of many related minimizations, or when the minimization is only a small intermediate step in a larger calculation.

A more recent development is that of interactive minimization, where the user can follow the progress of the search by means of a CRT screen or other output device, and modify the search procedure accordingly. Although this appears a potentially powerful technique, it is rather expensive in terms of real-time computer resources, and I am skeptical about its ultimate efficiency for two reasons:

i) Computer output devices and human geometric insight are both notoriously poor in spaces of high dimensionality, and cases of many variables are usually the only ones difficult enough to warrant the use of expensive computing techniques.

ii) If a human is able to direct a many-dimensional search more efficiently than a computer, it is probably a sign of deficiency in the numerical methods used by the computer rather than brilliant insight on the part of the human.

However, interactive minimization is still relatively new, and could prove very useful for some problems, including the development of new algorithms for off-line minimization programs.
REFERENCES

(References 18-31 are not referred to specifically in the text, but are added as useful general references.)


APPENDIX

SOME SAMPLE PROBLEMS FOR MINIMIZATION Routines

We assemble here a collection of test problems which found to be useful in verifying and comparing different minimization routines. Many of these are standard functions upon which it has become conventional to try all new methods, quoting the performance in the publication of the algorithm.

A1. Rosenbrock's curved valley

\[ F(x,y) = 100(y - x^2)^2 + (1 - x)^2 \]

start point: \( F(-1.2,1.0) = 24.20 \)
minimum: \( F(1.0,1.0) = 0 \).

This narrow, parabolic valley is probably the best known of all test cases. The floor of the valley follows approximately the parabola \( y = x^2 + 1/200 \), indicated by the dashed line in the diagram. In the cross-hatched area above the dashed line, the covariance matrix is not positive-definite. On the dashed line it is singular. Stepping methods tend to perform at least as well as gradient methods for this function.

A2. Wood's function of four parameters

\[ F(w,x,y,z) = 100(x - w^2)^2 + (w - 1)^2 + 90(z - y^2)^2 \]
\[ + (1 - y)^2 + 10.1[(x - 1)^2 + (w - 1)^2] + 19.8(x - 1)(z - 1) \]

start point: \quad F(-3,-1,-3,-1) = 19192
minimum: \quad F(1,1,1,1) = 0 .

This is a fourth-degree polynomial which is reasonably well-behaved near the minimum, but in order to get there one must cross a rather flat, four-dimensional "plateau" which often causes minimization algorithms to get "stuck" far from the minimum. As such it is a particularly good test of convergence criteria and simulates quite well a feature of many physical problems in many variables where no good starting approximation is known.


A3. Powell's quartic function

\[ F(w,x,y,z) = (w + 10x)^2 + 5(y - z)^2 + (x - 2y)^4 + 10(w - z)^4 \]

start point: \quad F(3,-1,0,1) = 215
minimum: \quad F(0,0,0,0) = 0 .

This function is difficult because its matrix of second derivatives becomes singular at the minimum. Near the minimum the function is given by \((w + 10x)^2 + 5(y - z)^2\) which does not determine the minimum uniquely.


A4. Fletcher and Powell's helical valley

\[ F(x,y,z) = 100[(z - 10\psi(x,y))^2 + (\sqrt{x^2 + y^2} - 1)^2] + z^2 \]

where \quad \[2\pi\psi(x,y) = \arctan \left(\frac{y}{x}\right) \quad \text{for} \quad x > 0\]
\[= \pi + \arctan \left(\frac{y}{x}\right) \quad \text{for} \quad x < 0\]

start point: \quad F(-1,0,0) = 2500
minimum: \quad F(1,0,0) = 0 .

\(F\) is defined only for \(-0.25 < \psi < 0.75\).
This is a curved valley problem, similar to Rosenbrock's, but in three dimensions.

[Reference: Comput. J. 6, 163 (1963).]

A5. Goldstein and Price function with four minima

\[ F(x,y) = (1 + (x + y + 1)^2 \times (19 - 14x + 3x^2 - 14y + 6xy + 3y^2)) \]

\[ \times (30 + (2x - 3y)^2 \times (18 - 32x + 12x^2 + 48y - 36xy + 27y^2)) \]

local minima:

- \( F(1.2, 0.8) = 840 \)
- \( F(1.8, 0.2) = 84 \)
- \( F(-0.6, -0.4) = 30 \)

global minimum:

- \( F(0, -1.0) = 3 \).

This is an eighth-order polynomial in two variables which is well-behaved near each minimum, but has four local minima and is of course non-positive-definite in many regions. The saddle point between the two lowest minima occurs at \( F(-0.4, -0.6) = 35 \), making this an interesting start point.

[Reference: Math. Comp. 25, 571 (1971).]

A6. Goldstein and Price function with many minima

\[ F(x,y) = \exp \left\{ \frac{1}{2} (x^2 + y^2 - 25)^2 \right\} + \sin^8 (4x - 3y) + \frac{1}{2} (2x + y - 10)^2 \]

global minimum:

- \( F(3, 4) = 1 \).

This function has "many" local minima.

[Reference: Math. Comp. 25, 571 (1971).]

A7. Quadratic function in four parameters

\[ F(x, y, z, w) = \frac{1}{70} (21x^2 + 20y^2 + 19z^2 - 14xz - 20yz) + w^2 \]
minimum: \[ F(0,0,0,0) = 0 \]
\[
\begin{pmatrix}
4 & 1 & 2 & 0 \\
1 & 5 & 3 & 0 \\
2 & 3 & 6 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

covariance matrix:

Except for the reasonably strong parameter correlations, this function poses no special problem to any minimization routine. But the author has found it useful in debugging programs based on quadratically convergent methods, since these programs should minimize the function exactly in one iteration. It is also used to check the calculation of the covariance matrix.

A variation consists of adding \(|x| - 1\) whenever \(|x| > 1\), and similarly with the other variables. This introduces in a reasonably smooth way terms which alter the quadratic behaviour far from the minimum while leaving it unchanged inside the unit cube, thus providing a test for those methods which are supposed to converge to the correct covariance matrix by updating.

A8. Chebyquad

\[
F(\mathbf{x}) = \sum_{i=1}^{n} \left\{ \int_{0}^{1} T_i(x') \, dx' - \frac{1}{n} \sum_{j=1}^{n} T_i(x_j) \right\}^2
\]

where \(T_i(x)\) are shifted Chebyshev polynomials of degree \(i\);

start point: \(x_j = j/(n + 1)\)

This function is designed to have a variable and possibly large number of parameters, and to resemble functions encountered in actual practice rather than being contrived to be especially difficult. Each term of \(F\) represents the squared difference between the true integral of a polynomial of degree \(i\) and the integral estimated by Chebyshev (equal-weight) quadrature on \(n\) points:

\[
\int_{0}^{1} P(x) \, dx \approx \frac{1}{n} \sum_{j=1}^{n} P(x_j)
\]
The starting values correspond to equally spaced points $x_j$, which is not too far away from the solution. Fletcher gives a complete Algol-coded procedure for this function in the reference quoted below.


A9. Trigonometric functions of Fletcher and Powell

$$F(\mathbf{x}) = \sum_{i=1}^{n} \left[ E_i - \sum_{j=1}^{n} (A_{ij} \sin x_j + B_{ij} \cos x_j) \right]^2,$$

where

$$E_i = \sum_{j=1}^{n} (A_{ij} \sin x_{0j} + B_{ij} \cos x_{0j}).$$

$B_{ij}$ and $A_{ij}$ are random matrices composed of integers between $-100$ and $100$; for $j = 1, \ldots, n$: $x_{0j}$ are any random numbers, $-\pi < x_{0j} < \pi$;

start point: $x_i = x_{0j} + 0.1\delta_j$, $-\pi < \delta_j < \pi$

minimum: $F(\mathbf{x} = \mathbf{x}_0) = 0$.

This is a set of functions of any number of variables $n$, where the minimum is always known in advance, but where the problem can be changed by choosing different (random) values of the constants $A_{ij}$, $B_{ij}$, and $x_{0j}$. The difficulty can be varied by choosing larger starting deviations $\delta_j$.

In practice, most methods find the "right" minimum, corresponding to $\mathbf{x} = \mathbf{x}_0$, but there are usually many subsidiary minima.

[Reference: Comput. J. 6, 163 (1963).]
FUNCTION PARAMETRIZATION

H. Wind
CERN, Geneva, Switzerland

1. GENERAL PRINCIPLES

Let us discuss some general principles and tricks for evaluating a reliable explicit expression of a function which is given at a number of points. Since we shall only deal with linear models, instead of "parameters", I would rather use the word "coefficients". Coefficients are used to represent a function explicitly. (The use of linear models does not imply approximating linear functions!) How to derive coefficients when the function is known at a finite number of points is the subject of this talk. We first consider the case of one variable. Let $y_j$ be given at $N$ reference points $x_j$. We can find intermediate values by linear interpolation. However, we might need a large number of reference points to achieve sufficient precision. The other extreme is to take all $N$ points into account with the Lagrange interpolation formula

$$y(x) = \sum_{j=1}^{N} \left\{ \prod_{i \neq j}^{N} \frac{x - x_i}{x_j - x_i} \right\} y_j.$$  \hspace{1cm} (1)

We notice that $y(x)$ is an $(N - 1)^{th}$ order polynomial in $x$ and that for $x_k$ we get $y_k$. We can consider Eq. (1) as a weighted average of the function values. Although Eq. (1) is an explicit expression for $y(x)$, it is a representation that is not very efficient to compute repeatedly; we need $N(N - 1)$ multiplications (not to speak of the divisions). It is therefore better to write Eq. (1) as

$$y(x) = \sum_{i=1}^{N} c_i x^{i-1}. \hspace{1cm} (2)$$

Now there are only $(N - 1)$ multiplications needed, e.g. for $N = 4$

$$y(x) = c_1 + x \cdot \left[ c_2 + x \cdot (c_3 + c_4 \cdot x) \right].$$
The coefficients are determined by the condition that
\[ y_j = \sum c_i x_j^{i-1}. \]

These are \( N \) linear equations in the \( N \) unknown coefficients. This set of
equations can readily be solved by matrix inversion. We consider
throughout these talks only those models where the coefficients appear
in a linear way.

Very often we want fewer coefficients \( (M) \) than we have observations
\( (N) \) and we then usually make a least squares fit. For a least squares
fit we minimize
\[ \sum_j \left( y_j - \sum_i c_i x_j^{i-1} \right)^2 \]
giving
\[ \sum_j y_j x_j^{k-1} = \sum_i \sum_j c_i x_j^{(k-1)+(i-1)} \quad \text{with} \quad k = 1, \ldots, M < N, \quad (3) \]
i.e. \( M \) linear equations. There are two reasons for minimizing the sum
of squares rather than, for example, the sum of the fourth powers. The
first reason is that, if there is a random error of measurement with a
normal distribution, the least squares solution is the most likely one\(^1\). The other reason is for convenience; making the fit for any other power
would be much more difficult.

There can be various purposes for making a least squares fit besides
the smoothing out of measurement errors. We may want a function repre-
sentation with as few coefficients as possible for a given precision. It
is also possible that the least squares solution is more reliable for
interpolation than the exact fit even in the absence of measurement errors.

Since both Eqs. (1) and (3) are unique, you may wonder why I am here.
Clearly, whether we use Chebyshev, Legendre, or any other sort of poly-
nomial, \( y(x) \) is the same. Let me therefore straight away tell you that
Eqs. (1) and (3) very often give the wrong answer. There are two
reasons why this is so, which may surprise you.
The first reason is rounding errors in computers. For this reason, in the case of Eq. (1), we should not take \( N \) more than 10 to 50, depending upon the function; when making a least squares fit, we run even more quickly into problems of finite machine precision\(^2\). Let us make a least squares fit using many equidistant points, \( N \gg M \). In that case, the ratio of the largest and smallest elements of the \( M \) by \( M \) inverted matrix derived from Eq. (3) is\(^3\)

\[
\left\{ \left( \frac{3M - 1}{2} \right)! \right\}^2 \left/ \left\{ M^3 \left( \frac{M - 1}{2} \right)! \right\}^6 \right. 
\]

For \( 7 < M < 15 \), within a factor 10, this happens to be equal to \( 10^M \). Consequently we cannot evaluate more coefficients than there are decimal places in the computer.

It is the combination of using equidistant reference points and polynomial interpolation that causes rounding errors to become so critical. Consider, for example, the polynomial

\[
y(x) = \prod_{i=1}^{12} (x - i) = x^{12} - 78x^{11} + 2717x^{10} + \ldots 
\]

(4)

This is, except for scaling factors, the only polynomial of degree 12 which has 12 equidistant zeros. If we would now change the term \( x^{12} \) to \( (1 + 10^{-6})x^{12} \) we would find that not only some roots have changed substantially but 4 roots have disappeared completely (from the real axis). If we go to higher order the effect of rounding errors grows dramatically. Let us consider

\[
y(x) = \prod_{i=1}^{20} (x - i) = \sum_{i=0}^{20} a_i x^i.
\]

When, for example, \( a_{12} \) is multiplied with \( 1 + 10^{-14} \) as many as 6 of the 20 zeros disappear from the real axis.

Acton\(^4\), in a similar context, remarks that you only need to see one tiger to know that you are in a jungle!

But even if we were not affected by machine rounding errors, the answer could be completely wrong and even get worse the more points we take. For example, consider the function
\[ y = \frac{1}{1 + 25x^2} \quad -1 \leq x \leq 1. \] (5)

It can be shown\(^5\) that by taking a sufficiently large number of equidistant reference points, the deviation between these points will grow indefinitely if \(|x| > 0.726\). I find the reason for this behaviour fascinating, and it is that this particular function has a singularity in the complex plane at \(x = i/5\)!

Just because this function misbehaves in the imaginary region it cannot be accommodated in the real world. Who could guess beforehand what an innocent calibration curve looks like in the complex plane. When I mentioned this so-called Runge phenomenon at Amsterdam, someone mentioned "convergence circle". But it is not as simple as that. For example, for \(-1 \leq x \leq 1\) the fit of the function \(1/(1 + 2x^2)\) does improve as more and more equidistant points are taken. These two cases are distinguished by the criterion whether the singularity in the complex plane lies within an ellipse-like curve going through \(z = +1, z = -1, z = +0.525525\ldots i\) and through \(z = -0.525525\ldots i\).

Let us go back to our first question. What made us believe that polynomial interpolation would give the correct answer? A basic concept of science is that nature is simple, so we accept the most simple model consistent with the observations. Consequently, we consider the model

\[ \sum_{i=0}^{N} c_i x^i \] (6)

more likely to correspond to reality than, for example,

\[ \sum_{i=10}^{N+10} c_i x^i . \]

But is model (6) really more simple than say

\[ \sum_{i} c_i \cos ix , \]

and would local linear interpolation represent a simple model? Well, you might argue that this represents a function with a discontinuity in
Fig. 1  An example of polynomial interpolation
the first derivative. If we have reason to believe that the function
is continuous we would like to make use of this additional *a priori*
knowledge by using a model which is also continuous. To insist, however,
that a function is continuous in all higher order derivatives is an
extremely severe condition. For example, Fig. 1 shows the only 9th order
polynomial going through the 10 points indicated; the effect of machine
precision can be neglected. Intuitively, one feels that the simplest
function going through the points should not look as shown, for example,
because the radius of curvature is too small.

So let us choose another model where we make the concession not to
insist that all derivatives be continuous, but instead minimize the
radius of curvature. Then we get the curve shown in Fig. 2, which is a
cubic spline fit. Here, the third derivative is allowed to be discon-
'tinuous at the reference points and \( \int y''^2 \, dx \), which is approximately the
integrated inverse radius of curvature squared, is a minimum. (The exact
value of the integrated curvature squared is \( \int y''^2/(1 + y'^2)^{3/2} \, dx \)). Such
a spline fit is often more reliable than a regular polynomial interpola-
tion and one can also use a spline-model to make a least squares fit. In both cases, there can be a saving in computing time when the continuity constraints are incorporated in the model rather than added to them. Consider a simple example, Fig. 3. Let us assume we have six points given in three intervals.

In the first interval we assume a straight line, in the second a parabola, and in the third interval again a straight line. We thus have $2 + 3 + 2 = 7$ coefficients to determine, and we further require continuity of the function and its first derivative; this imposes $2 + 2 = 4$ constraint equations. The least squares criterion thus gives 7 equations plus 4 constraints, requiring the inversion of an 11 square matrix. However, it is much better to consider only the three coefficients in the 2nd interval "free" and define the function in the other two intervals from this by extrapolating its end values' slopes. In this case we only have three equations, giving a 3 square matrix to invert (see Appendix 1).

So we see that it is very advantageous to incorporate the constraints in the model; instead of being added $(7 + 4)$ they are subtracted $(7 - 4)$.

Fig. 3  An example of a spline fit
A more dramatic example of this is the case of fitting magnetic field observations.

Let us assume that we have made observations of the three components of a magnetic field. Suppose we want to determine an explicit polynomial expression in $x$, $y$ and $z$ of the three components, such that we exclude terms with a sum of powers in $x$, $y$ and $z$ larger than $n = 10$. Then there are $(n + 1)(n + 2)(n + 3)/2 = 858$ coefficients to be determined and there are $n(n + 1)(n + 3)/2 = 715$ constraint equations due to Maxwell's equations. Adding the constraints, we would have to invert a 1573 square matrix; incorporating the constraints leads to the inversion of a 143 square matrix. (Separation into symmetry classes can further save about a factor 8, and we will come back to this later.)

We have discussed the spline model as an alternative to a regular polynomial model. There is another alternative, which is the use of trigonometric functions, which has important advantages over a polynomial model. In particular, we can be sure to get closer to the real function as more and more equidistant points are taken within the same interval. As we saw, with polynomials this is not always true. Before mentioning other advantages of a trigonometric model, let us deal with the obvious disadvantage: the implicit assumption of periodicity. If we were to make an ordinary Fourier analysis on a non-periodic function, the discontinuity at 0 or $2\pi$ would result in a convergence of the coefficients of

$$a_i \propto 1/i.$$ 

Consequently, one would need in the order of a thousand coefficients, and at least as many points, to achieve a precision of 1 in 1000. This can be overcome by considering the function to be fitted as running within the interval $[0, \pi]$; the interval $[\pi, 2\pi]$ is then reserved for smooth transition into the neighbouring period.

For example, we can make a fit like $f(x) \approx \sum a_i \cos ix$ for $0 < x < \pi$, thus implicitly assuming symmetry about $x = 0$ and about $x = \pi$ (see Fig. 4). The imposed discontinuity in the derivative leads to a convergence of the coefficients of

$$a_i \propto 1/i^2.$$

Fig. 4  To make a fit in trigonometric functions, we need to fit the end points first.
The above convergence is not sufficient, in general, and a possible improvement is first to subtract a straight line. An alternative method can be adopted, namely to use the two cosine functions $\cos (0 \cdot x) = 1$ and $\cos x$ for making the end points zero:

$$f(x) = \frac{f(0) + f(\pi)}{2} + \frac{f(0) - f(\pi)}{2} \cos x + \sum_{i} a_i \sin ix .$$

The implicit assumption now is that, after making the end points zero, the function is asymmetric. The second derivative has now a discontinuity at the end points, which leads to a convergence as

$$a_i \propto 1/i^3 .$$

Another improvement would be to make the derivatives zero at the end points by two sine functions. The model then becomes

$$f(x) = \frac{f'(0) - f'(\pi)}{2} \sin x + \frac{f'(0) + f'(\pi)}{4} \sin 2x + \sum a_i \cos ix .$$

The first discontinuity is now in the third derivative, which leads to a convergence (always for large $i$) as

$$a_i \propto 1/i^n .$$

Estimating the function value and the second derivative at the end points leads to $a_i \propto 1/i^5$, and so on. Note that we use alternatively sines and cosines.

In practice, it is usually not worth while to go beyond the case with convergence of $a_i \propto 1/i^3$. The number of terms is then likely to be determined by the function itself rather than by the imposed discontinuity in the 2\textsuperscript{nd} derivative. This can also be formulated by stating that the derivatives at the end points cannot be well determined.

We compare this with spline fits. Whereas here we fit a function which we assume to be discontinuous (in higher order) with a continuous model, for splines we fit a continuous function with a discontinuous model.

Another way of looking upon this is as a fit "in stages". We first deal with the end points and we treat what is then left with a model which does not affect the end points.
Fitting in stages can be very useful. For example, it can be used to solve a three-dimensional problem as a threefold two-dimensional problem. We will return to this in the application of fitting magnetic field data.

A very common fit in stages is the separation into symmetry classes. We thus first fit a function with a symmetric model and then fit the residue with an asymmetric model. In particular, this offers an advantage when the function to be fitted is dominantly either symmetric or asymmetric. We thus avoid only every other term becoming significant (or only one out of eight in a three-dimensional problem).

At this stage, we must try to recall our main line of thought. We saw that polynomial interpolation for equidistant reference points is very risky. We considered splines and trigonometric functions as alternative models. Both models are guaranteed to converge to the real function, as opposed to a polynomial model. For the trigonometric model we had to do something to avoid problems due to the implicit periodicity. We will now discuss another technique to achieve this. We assume again that we have \( y_j \) given at points \( x_j \) with \( -1 \leq x_j \leq 1 \). We now make a transformation (see Fig. 5).

\[
x = \cos \theta, \quad y(x) = f(\theta).
\]

Then \( \frac{\partial^{2n-1} f(\theta)}{\partial \theta^{2n-1}} \) is zero at \( \theta = 0 \) and at \( \theta = \pi \). This is the ideal function to expand in cosine series:

\[
f(\theta) = y(\cos \theta) = \sum a_i \cos i \theta.
\]

Transforming back to \( x \) we get as a model

\[
y(x) = \sum a_i \cos (i \arccos x).
\]

This is a polynomial expansion, since

\[
\cos (i \arccos x) \equiv T_i(x),
\]

where

\[
T_0(x) = 1, \quad T_1(x) = x
\]

\[
T_{n+1}(x) = 2T_n(x)T_1(x) - T_{n-1}(x)
\]

are the Chebyshev polynomials (of the first kind).
So there we are; back to polynomials! Why can polynomials now be used? To understand this, we have to discuss orthogonality. If we compare using an orthogonal model with reading a newspaper, then using a

\[ F(x) = f(x) = \sum a_i \cos \theta = \sum a_i \cos \left( \arccos x \right) = \sum a_i T_i(x) \]

**Fig. 5** The relation between Fourier analysis and a Chebyshev expansion
non-orthogonal model is like reading a computer manual. With a newspaper, you can pick any article and collect more or less valuable information. With a computer manual, you must, to understand one page, have understood all previous pages as well as all the following pages. This also holds for a non-orthogonal model. To evaluate the value of one coefficient, you have to know all other coefficients, preceding as well as following; i.e. you have to invert a matrix. For an orthogonal model, on the contrary, all coefficients can be evaluated separately; the matrix to be inverted is diagonal.

Orthogonality in this context depends on the reference points:

$$\sum_{\alpha} f_i(x_{\alpha}) f_j(x_{\alpha}) = \delta_{ij} \sum_{\alpha} f_i^2(x_{\alpha}) ,$$  \hspace{1cm} (8)

where $f_i(x)$ are the functions used in the model

$$y(x) = \sum_i a_i f_i(x) .$$

It is not only for convenience that we prefer an orthogonal model. It is a safe way to avoid machine-rounding errors becoming relevant. But, as you remember, machine precision was only one of the reasons for failure of equidistant polynomial interpolation. However, using an orthogonal model also solves that problem in most cases.

To satisfy the orthogonality condition, we can either choose the model $f_i(x)$ or choose the reference points $x_{\alpha}$ appropriately. To have good convergence, we would like to have a model where $f_i(x)$ fluctuates between $\pm 1$ \(^a\). The ordinary monomials $f_i(x) = x^{i-1}$ for $-1 \leq x \leq 1$ satisfy this requirement, but it is not possible to find $x_{\alpha}$ such that the orthogonality condition is satisfied.

Trigonometric functions fluctuate between $\pm 1$, and if $x_{\alpha}$ is equidistant, the orthogonality condition (8) is satisfied. So, that is why we got the right answer when using a trigonometric model for equidistant points. Chebyshev polynomials

$$T_i(x) = \cos (i \arccos x)$$

also fluctuate between $\pm 1$. 
To satisfy the orthogonality condition, we now want to choose \( x_\alpha \)
such that \( \theta_\alpha = \arccos x_\alpha \) is equidistant (see Fig. 5). To be precise, we choose

\[
x_\alpha = \cos \left( \frac{(2\alpha - 1)\pi}{2N} \right),
\]

(9)

where \( \alpha = 1, \ldots, N \). For these points \( T_N(x_\alpha) = 0 \). Taking these reference points for the original function

\[
y = 1/(1 + 25x^2)
\]

leads to good results when we make polynomial interpolation. We found, however, that if we use the above "Chebyshev points" with a "Fourier model" the results are at least as bad as using equidistant (i.e. Fourier) points with a polynomial model\(^9\). "Bad" here can mean wrong by orders of magnitude and, as more terms are taken, the error grows bigger and bigger. Similarly as when reading a computer manual, your ignorance seems to grow bigger and bigger. Concluding we state that if you are given equidistant points you should use a trigonometric model and if you can choose your points, choose them "Chebyshev-wise" [Eq. (9)] and use a Chebyshev model. As mentioned above, a property of Chebyshev polynomials is that \( T_i(x) \) has \( i + 1 \) extrema for which \( |T_i(x)| = 1 \) for \(-1 \leq x \leq 1\). This property can also be looked upon as the definition. It can be shown to follow from this that

\[
\max_{-1 \leq x \leq 1} |T_n(x)| \leq \max_{-1 \leq x \leq 1} |L(x)|,
\]

where \( L(x) \) is any other polynomial of degree \( n \) with the same leading term as \( T_n(x) \). We compare this with Legendre polynomials which have the property

\[
\frac{1}{1} \int_{-1}^{1} p_n^2(x) \, dx < \int_{-1}^{1} L^2(x) \, dx,
\]

where \( L(x) \) is again any other polynomial of degree \( n \) and the same leading term as \( P_n(x) \). For example

\[
T_3(x) = 4x^3 - 3x
\]
giving

\[
T_3(\pm 1) = \pm 1, \quad T_3(\pm \frac{1}{2}) = \pm 1.
\]
Comparing this with, for example, the third Legendre polynomial
\[ \frac{8}{5} P_3(x) = 4x^3 - 2.4x, \]
we get extrema ±1.6 and ±0.716 for \( x = \pm 1 \) and \( x = \pm \sqrt{1/5} \). On the other hand, let us compare
\[ \int_{-1}^{+1} T_3^2(x) \, dx = 0.971 \]
with
\[ \int_{-1}^{+1} \left( \frac{8}{5} P_3(x) \right)^2 \, dx = 0.731. \]

Chebyshev polynomials do not suffer so easily from rounding errors as polynomials with equidistant zeros [see Eq. (4)].

Consider, for instance,
\[ y(x) = T_{12}(x) = 2048x^{12} - 6144x^{10} + \ldots. \]

If we now change the term \( 2048x^{12} \), not by \( 10^{-6} \) as in Eq. (4) but to \( 2049x^{12} \), the position of the zeros does not change a great deal and we would have to increase it to \( 2049.6x^{12} \) before any root would become complex.

Suppose we approximate an \( n^{th} \) order polynomial with an \( (n - 1)^{th} \) order Chebyshev polynomial expansion and compare this with an \( (n - 1)^{th} \) order Legendre polynomial expansion. For Chebyshev we have the minimum maximum error ("minimax"), for Legendre the minimum mean square error. My experience is that the first is easier to sell. The reason for this is similar to the reason for taking out an insurance: on the average you lose, but the risk of going very wrong is smaller. In the first case one would like to choose the reference points \( x_\alpha \) such that \( T_N(x_\alpha) = 0 \) and for the second such that \( P_N(x_\alpha) = 0 \). These latter points are also those used for Gauss integration. As indicated in Fig. 6, there is not a large difference between these two sets of \( x_\alpha \). This rule may suggest that the orthogonality condition (8) is fulfilled in both cases. This is not quite true. The equation that does hold is
\[ \sum_\alpha w_\alpha f_i(x_\alpha) f_j(x_\alpha) = \delta_{ij} \sum_\alpha w_\alpha f_i^2(x_\alpha), \]
Fig. 6 The zeros of Chebyshev and of Legendre polynomials are suited as reference points; there is not much difference between the two.

where indeed $w_\alpha = 1$ for the Chebyshev case but for the Legendre case $w_\alpha$ are equal to the weights used for Gauss integration.

So, in general, if we want to measure a function, we should choose the reference points more to the border of the interval in which the function is to be known. On second thought, this is not surprising. One would expect good precision at a point if there is about the same number of reference points on each side of that point, or alternatively if there are close reference points. One wonders to what extent this is true in other fields. Take pattern recognition. On first thought you would think that typical cases provide the best reference but actually atypical cases are more suited.

The inherent assumption here is that there are enough observations for the (Chebyshev- or Legendre-) expansion to converge.

As we shall see, the computation of reference trajectories through a magnetic field can be very profitable. For these tracks, we would rather choose those which are marginal than the typical tracks. The "Chebyshev distribution" is like $1/\sqrt{1 - x^2}$, which looks very different from a normal distribution $\exp(-x^2)$.

We now come to the problem concerning the choice of the most efficient representation. I have been advocating the use of trigonometric functions, Chebyshev and Legendre polynomials. You might think that computing a sum

$$\sum_{i=1}^{N} a_i f_i(x),$$

(10)
where \( f_i(x) \) is one of the above functions, takes much more computing time than the series

\[
\sum_{i=1}^{N} a_i x^{i-1}
\]

which, as we saw, needs only \( N - 1 \) multiplications. To make a sum (10), however, also takes only \( N \) multiplications as a result of the recursive relation [Eq. (7)] between the functions \( f_i(x) \).

We take as an example the sum of a Chebyshev series. Then

\[
\sum_{i=0}^{N-1} a_{i.1} \cdot T_i(x) = \sum_{i=0}^{N-2} a_{i.1}^{*} \cdot T_i(x)
\]

where

\[ a_{i}^{*} = a_{i} \quad \text{for} \quad i < N - 3 \]

\[ a_{N-3}^{*} = a_{N-3} - a_{N-1} \]

and

\[ a_{N-2}^{*} = a_{N-2} + a_{N-1} \cdot \beta, \quad \text{where} \quad \beta = 2x. \]

This can readily be verified with Eq. (7).

We see that, by one multiplication, the number of terms is reduced by one, so that only \( N \) multiplications are needed for the above sum (\( \beta \) needs only to be determined once; for a Fourier sum one trigonometric function evaluation has to be added). To avoid confusion, let me remark that this has nothing to do with the so-called Fast Fourier Transform.

There is still a problem of how to divide an interval, over which a function is known, into subintervals. If this interval is divided into two independent subintervals, the total number of coefficients goes up but the computing time needed to evaluate the function is lower since only one set of coefficients is used at a time. The price one pays for evaluating the function could be the computer space-time product; so this is what we want to minimize. I have tried very many functions of all sorts and sizes and there seems to exist an easy rule of thumb independent of the nature of the function. The rule is that the interval, over which a function is to be defined, needs to be divided into so many subintervals that per interval \(-\ln E\) coefficients are needed to obtain a prescribed relative precision \( E \). So, for an accuracy of \( 1\% \) we want
about seven coefficients per interval. As you can see from the plots (Fig. 7) this number is not very critical, but it does indicate on the one hand that linear interpolation is too space consuming and, on the other hand, that representing a function by very many coefficients is too time consuming. This rule also avoids problems due to machine rounding errors.

There is still one point which I really should have said in the beginning. This is the elimination of wrong points. What often happens is that in a series of measurements, there is a point entirely wrong as a result of a misprint, a wrong tape reading, a wrong label, etc. These sorts of errors must be eliminated before making a fit. If not, you start with one point wrong but after the fit all points are wrong.

2. APPLICATIONS

We have mentioned several procedures for making a fit. Before discussing some applications we will summarize them briefly.

1) Wrong points should be eliminated.

2) If the reference points are equidistant, a trigonometric model is preferable to a polynomial model.

3) A large radius of curvature is often a better criterion for smoothness than continuity in all higher order derivatives.

4) Constraints (a priori knowledge) should be incorporated in a model rather than be added afterwards (with Lagrange multipliers).

5) A polynomial model for equidistant points can give entirely wrong results, even if there were no problem of machine rounding errors.

6) To minimize the maximum error Chebyshev polynomials should be used, but to minimize the average error Legendre polynomials are indicated. The \( n \) reference points are then to be chosen such that (for \( \alpha = 1, \ldots, n \)) \( T_n(x_\alpha) = 0 \) or \( P_n(x_\alpha) = 0 \), respectively.

7) The number of multiplications needed to evaluate a Chebyshev, Legendre, or Fourier sum need not be more than the number of coefficients.
Fig. 7  The functions shown on the left are divided into subintervals. On the right the computer space time product is plotted as a function of the number of coefficients per subinterval needed to achieve a given relative precision. The arrows indicate the minimum predicted by an easy rule of thumb.
8) To make a fit in sine functions one can use cosine functions to make the end points (and maybe even order derivatives) zero. To make a fit in cosine functions, one can use sine functions to make odd order derivatives zero at the end points. These end points in both these cases are assumed to be 0 and \( \pi \).

9) One can often simplify a fitting problem by fitting in stages; the separation into symmetry classes is an example of this.

10) To minimize the computer space-time product, the interval of interest of a function should be divided up into subintervals such that about \(-\ln E\) coefficients per subinterval are required to achieve a relative precision \( E \) (empirical law).

As a practical example, we will now discuss very briefly the problem of processing magnetic field data. We assume that the volume in which we want to know the field is empty so that the Maxwell equations \( \nabla \cdot \mathbf{B} = 0 \) and \( \nabla \times \mathbf{B} = 0 \) hold.

It is possible to design a polynomial model, in Cartesian coordinates, that obeys implicitly the above constraints\(^{11}\). However, we cannot choose the reference points such that this model is an orthogonal one. The number of coefficients which one can evaluate is consequently limited by machine rounding errors. Another disadvantage is the rather complicated form of this, so-called, harmonic polynomial model. Both these disadvantages disappear if we use a trigonometric model. The trigonometric model published\(^{12}\) deals with one component which is constrained by the condition \( \Delta B_1 = 0 \) (zero Laplacian).

A function with zero Laplacian is determined by its boundary values. A magnetic field component need therefore only be measured on the boundary of the (empty) volume in which this component is to be known. Moreover, a function with zero Laplacian has its maximum at the boundary. If boundary observations are fitted with a model obeying this constraint, the difference between the real and the fitted function, also being harmonic, must reach its maximum at the boundary. The boundary observations are therefore the best choice of reference points\(^{13}\). Let us therefore assume that we have measurements of a magnetic field component at the boundary of a rectangular box and that we make a fit in trigonometric functions.
In order to make a fit in stages, we would prefer the fit in one plane to be made independently from the fit in the planes perpendicular to this plane. Consequently for the above problem, we make the bulk fit in sine functions and use a few cosines to fit the values at the edges.

This method is now used regularly at CERN. For instance the large $\Omega$ magnet has been measured at 17,684 boundary points. Measuring throughout the volume would require 143,472 points. Not only has a lot of extra work been saved but the interior values are more accurately known than could have been achieved by direct measurement, because the computed values are obtained by a least squares fit with a certain redundancy.

Let us now consider a slightly different model. Here, we make the bulk fit with cosines and use sines to make the derivatives at the edges zero. This model is particularly suited to accommodate boundary measurements of those components of a magnetic field which are perpendicular to the boundary surface (which is again assumed to be a rectangular box) [see Appendix 2 \textsuperscript{1b}].

We can design many other models with built-in constraints, but let us now rather tackle the far more important problem of processing spark chamber data. In view of modern high data taking rates one can in a relatively short period save or waste hundreds of hours of computing time in this field. Processing spark chamber data can usually be divided into qualitative and quantitative analysis. The first may deal with labelling each spark coordinate according to which track it belongs to. In the quantitative analysis, we may want to determine, for example, the value of the momentum. Here we will deal with the quantitative analysis only and with single tracks. One method of doing this is to estimate a value of the momentum, compute a trajectory through the known magnetic field, and adjust the momentum until the trajectory coincides sufficiently with the observations.

If we have a large number of tracks to analyse, this takes far too long and moreover one often stores the complete magnetic field map which is very space consuming. In this case (of very many events or tracks) it becomes worth while to evaluate the momentum and other quantities of interest as an explicit function of the observations. In practical cases,
up to five orders of magnitude in the computer space-time product can be saved\textsuperscript{15}). We thus use a number of reference tracks to determine this explicit expression.

Before we discuss this multi-dimensional fitting process, we have to reduce the observed track coordinates to the essential minimum. A particle follows a (one-dimensional) path in six-dimensional phase space. These paths trajectories have therefore five degrees of freedom. This means that, in general, we have to define \textit{at most} five functions of \textit{at least} five variables each. From now on, however, we will assume that there is only one quantity we are interested in and we will call this quantity the momentum $P$. To derive the other quantities, we follow the same procedure.

Let a track be given by $M$ coordinates ($M \geq 5$); an observed track can thus be considered as a point in $M$-dimensional space. Since there are only five degrees of freedom only five track coordinates would suffice, if it were not for the error of measurement. Because of the errors of measurement, we do not want to simply throw away the other $M - 5$ coordinates. Instead, we would like to determine five new coordinates which contain all the essential information. On the one hand, we thus hopefully smooth out errors of measurement and on the other hand fitting $P$ now takes place in five- rather than in, for example, 200-dimensional space (for the $\Omega$ project, there are about 100 plane detectors giving two coordinates per plane).

We can look upon the $N$ points (tracks) in $M$-dimensional space as lying in a five-dimensional subspace. We know that such a subspace must exist but defining it is another matter. It is, in fact, the search for this five-dimensional subspace that constitutes the heart of this problem. To help to define this subspace we study $N$ simulated tracks with the purpose of establishing a new coordinate system.

We can adopt two philosophies. One is that we use all our knowledge of the experimental layout to postulate a relation between the $M$ observed coordinates $\vec{x}$ and the five new coordinates $\vec{\xi}$. The other is to ignore this \textit{a priori} knowledge but instead fit a five-dimensional hyperplane through $N$ points in $M$ dimensions. We thus define a linear relation

$$\xi_j = \sum_i w_{ij} x_i .$$ (11)
This latter approach is, however, restricted to the five-dimensional subspace being a linear one.

In general it is not possible to define exactly the five-dimensional subspace by a convenient transformation. This is not due to errors of measurement; we can work with computed tracks without observational errors.

Let us list the properties which we would like the new variables $\xi$ to have, not knowing the exact transformation.

i) **Good description of the track.**

Ideally one should not only be able to derive $\xi$ from $\bar{x}$ but also $\bar{x}$ from $\xi$ by a simple inverse transformation. If this latter condition is fulfilled, we know for sure that no information is lost. We must be aware, however, that if this latter condition is not fulfilled, it does not necessarily mean the loss of relevant information.

ii) **The range of interest of one component of $\xi$ should be independent of the range of interest of the other four components.**

This makes it likely that every set of values of the new coordinates, each within its range, represents a possible track, so that we do not define the momentum function over a wider range than we are interested in.

iii) **The new variables should allow a good determination of $P$.**

As an example, let us assume that the function value $P$ is the horizontal component of the momentum. In that case, we could choose as one of the new variables an estimated value of the inverse of the radius of curvature in a horizontal plane.

iv) **It should be possible to compute, for a given $\xi$, the corresponding value of $P$ before making the fit.**

When computing a track we can determine $\bar{x}$, and therefore $\xi$, for given values of the momentum $P$. However, to have an optimum selection of reference tracks we would like to choose the values of $\xi$ as points suitable for Chebyshev interpolation. Knowing a function ($P$ in our case) at these points ensures an easy and reliable fit, in five dimensions just as well as in one dimension.
v) The new coordinates should have a computationally simple relation with the observed coordinates and also with \( P \).

Once \( P \) is known as an explicit function of the new coordinates we want to evaluate this function for observed new points (by contrast to the \( N \) reference points). The efficiency of the present method will thus be determined by the computer time (and space) for performing this operation.

Except for a simple set-up\(^{15}\), in general we cannot satisfy all the above five requirements [although if (i) is true then clearly (iv) must be true otherwise no analysis would be possible at all]. To satisfy condition (i) (i.e. \( \xi \rightarrow x \)) we need a model of a trajectory. This model would ideally have five parameters which can be used as the new coordinates. For reasons discussed earlier, a spline fit would seem an appropriate model. If the detectors are parallel this leads to a linear transformation from the original (observed) coordinates into the new coordinates (parameters) (see Appendix 1). But if such a transformation exists at all, one can find this by looking at a sufficiently large number of observed tracks, ignoring any a priori knowledge. We "simply" fit a five-dimensional hyperplane through the \( N \) points in \( M \) dimensions; this can readily be done for \( M \) up to a few hundred and for any \( N \) (see Appendix 3). By making this fit, the above conditions (i) and (ii) are satisfied as well as possible with a linear transformation. If the detectors are not parallel, it is very difficult to design a model in which the relation between the parameters and the observations is linear. When making the fit in \( M \)-dimensional space it was found that for non-parallel detectors a five-dimensional linear subspace fits noticeably worse than for parallel detectors, see Fig. 8. This indicates that it is not because of lack of ingenuity that one cannot design a good model, apparently there does not exist a model in which the coefficients appear in a linear way. I would therefore recommend having the detectors parallel or radiating from a common axis. In the latter case, the detectors are parallel after a simple coordinate transformation.

The first condition (i.e. \( \xi \rightarrow x \)) can in general not be satisfied by a five-dimensional linear transformation, but we can take a slightly higher dimensional subspace. In fact, the original \( M \) coordinates can be
transformed into $M$ new coordinates which have decreasing ranges, see Figs. 9 and 10. In Fig. 10 we see a levelling-off in these ranges: This is due to a (simulated) error of measurements. We can thus detect real or simulated measurement errors without any knowledge of the experimental set-up. Let us assume, for a moment, that we would need seven new variables to reconstruct $\vec{x}$ from $\vec{\xi}$ satisfactorily, i.e. within the error of measurement. We then still have $M-7$ new coordinates which are insignificantly small for all $N$ tracks considered. These latter coordinates therefore give constraints which can conveniently be used to test whether a new event is real, i.e. belongs to the same group as the $N$ reference tracks. In fact the sum of the squares of the $M-7$ co-ordinates equals the square of the distance to the seven-dimensional linear subspace in which all $N$ tracks lie.

The sixth and seventh new coordinates in our example are supposedly not insignificantly small (i.e. indistinguishable from the error of measurement), but they must be a function of the first five new coordinates because there are only that many degrees of freedom. By making this fit, i.e. $\xi_6(\xi_1, \xi_2, \xi_3, \xi_4, \xi_5)$ and $\xi_7(\xi_1, \xi_2, \xi_3, \xi_4, \xi_5)$ we thus
Fig. 9  Another example of the effect of introducing new variables

Fig. 10  An example of the distance to the M-dimensional subspace levelling-off as a result of errors of measurement
derive two more constraint equations. Concluding we state that for a newly observed track with M coordinates, we can derive five new coordinates, which can be used to express the momentum P, and M - 5 constraint equations. These constraint equations can be conveniently used for the qualitative analysis.

For the programmer it is very tempting to ignore a priori knowledge. One, thus, only needs a tape with a series of processed tracks and use it without feeding in the magnetic field, the detector positions, etc. There may, however, be some difficulties when using the final expression for actual observations. What if we have some coordinates missing? In practice not all detectors are always hit and even if hit may not fire. We have also mentioned that we may need to drag along more than five new coordinates, especially when the detectors are not parallel. If we have many detectors we can generate missing points by local interpolation. Local interpolation has also successfully been used for simulating parallel detectors.

If we do use a priori knowledge and choose sophisticated new variables, we are no longer restricted to linear transformations and we can hope to satisfy the conditions (i) to (v) better than we could with a linear transformation. For example, in the case of non-parallel detectors, the relation between the spline coefficients and the observations is non-linear. In this case we make a fit (i.e. invert a matrix) for each track observed.

Let us assume that we now have reduced the number of variables to five. We still have the problem of fitting the momentum function P. The value of this momentum function is known at the N points in five-dimensional space. If we are lucky enough to have Chebyshev reference points, making this fit is easy.

In one dimension the coefficients $c_j$ of a Chebyshev expansion

$$P(x) = \sum_j c_j T_j(x)$$  \hspace{1cm} (12)

are obtained by
\[ c_j = \sum_{\alpha=1}^{n} P(x_{\alpha}) T_j(x_{\alpha}) / \sum_{\alpha=1}^{n} T_j^2(x_{\alpha}) \]

\[ = \frac{2}{n} \sum_{\alpha=1}^{n} P(x_{\alpha}) \cos \left\{ j \pi (2\alpha - 1)/(2n) \right\} \quad j < n \]

(except for \(c_0\), which is half the above value), where \(P(x_{\alpha})\) is the function value at the "Chebyshev points"

\[ x_{\alpha} = \cos \left\{ (2\alpha - 1) \pi/(2n) \right\}. \]

For a five-dimensional fit we have

\[
P(\xi_1, \xi_2, \xi_3, \xi_4, \xi_5) = \sum_m \left[ \sum_{i} \sum_{j} \sum_{k} c_{ijk} \left\{ T_i(\xi_1)T_j(\xi_2)T_k(\xi_3) \right\} T_m(\xi_5) \right].
\]

The expression in \([\cdot\] brackets can be considered as the unknown coefficients of a one-dimensional fitting problem in \(\xi_5\) for fixed values of \(\xi_1\) to \(\xi_4\). Having solved this for all values of \(\xi_1\) to \(\xi_4\) we fit to the \(\xi_4\) dependence, i.e. the expression in \({}\) brackets, for every \(m\) and all values of \(\xi_1\) to \(\xi_3\), and so on.

Evaluating the coefficients in this case is fast and accurate; we could evaluate thousands of coefficients without problems of rounding errors or storage capacity. To have then an efficient representation with, for instance, one hundred coefficients we can simply take the largest hundred of these coefficients. A disadvantage of this method is that we might need very many reference points. If we would have, for instance, 10 points in each variable, we would have \(10^5\) trajectory computations. Well, it is not quite as bad as that; because of the five variables we will find two or three which vary over such a small range that only a few reference points in these variables are needed. On the other hand, for the variables which vary over a wide range we may well need more than 10 reference points. In practice we need several thousand
reference tracks, but the computing time (maybe hours) to generate these will pay off when we have a sufficiently large number of new tracks to process in the actual experiment. The great advantage of using Chebyshev points is that we are sure to end up with a very efficient representation in terms of the five new variables. The disadvantage, besides the great number of reference points needed, is that in general we cannot choose our reference tracks and then evaluate P other than by a lengthy iterative procedure. Moreover, these computations may already have been preceded by many Monte Carlo generated tracks to decide what the new variables are.

If we have not chosen our reference point in a convenient way but have obtained them from, say, Monte Carlo generated tracks, we can still make a fit but this is now more tricky. The problem then is: given N points scattered in five-dimensional space and with each point a function value P, how can one derive an explicit expression of P?

Before we discuss the strategies of multidimensional fitting let us estimate how big N (the number of points) ought to be. At first glance you might think that 500 points with five coordinates (and a P value) each, would be a fair amount.

However, there is a lot of space in five dimensions! If one would have 500 points scattered in five-dimensional space, there would be only a few interior points (points not lying on the convex envelope); they barely fill the volume. My guess is that one would need of the order of a million points to have at least half the points not on the boundary. In fact if the suggestion of taking more points near the boundary is followed up, then the fraction of interior points is even lower.

It is therefore surprising that in actual practice we can make a reliable fit with only a few hundred reference points. In the case of a magnetic field model we only need boundary observations, but this is because the function fitted has zero Laplacian. However, for spark chamber data there is not such a constraint. Well, there is of course Liouville's theorem but that would only impose a constraint if time should be one of the observed coordinates.
To make the fit in five-dimensional space, we have to choose a model. We have seen the advantages of spline models. It is probably possible to design a five-dimensional spline model with all continuity constraints built-in. However, this is (as far as I know) not yet available and we must use a more conventional model like Chebyshev polynomials (a trigonometric model would not be practical since we have no specific end points to fit). We thus write

$$P(\xi) = \sum_{jkkmn} c_{jklmn} T_j(\xi_1)T_k(\xi_2)T_l(\xi_3)T_m(\xi_4)T_n(\xi_5).$$ (14)

If we "pack" the five indices into one, we write this as

$$P(\xi) = \sum_i c_i f_i(\xi),$$

where $f_i(\xi)$ is the above product of five polynomials.

Written in this form, we can make the least squares fit by matrix inversion

$$\hat{\mathbf{c}} = (\mathbf{F}'\mathbf{F})^{-1}\mathbf{F}'\mathbf{F},$$

where

the $i$th element of $\hat{\mathbf{c}}$ is the coefficient $c_i$,

the $j$th element of $\hat{\mathbf{F}}$ is the known value of the momentum function $P(\xi_j)$ and $\hat{F}_{ji} = f_i(\xi_j)$.

The difficulty lies here in the maximum value of $i$. If we were, for instance, to include all possible combinations of $j$, $k$, $l$, $m$, and $n$, with each having a value between 0 and 5, we would have $6^5$ functions. To evaluate that number of coefficients we would need to invert a $6^5$ square matrix, the model being non-orthogonal. In the final expression, however, we are satisfied with, say, one hundred coefficients. To invert a $100 \times 100$ matrix is not too difficult (if you have a computer that is), but the number of ways one can take a hundred functions out of $6^5$ is abominable ($7 \times 10^{230}$ to be more precise). Clearly we need a strategy to select the (hundred) most suited functions. The obvious selection procedure is to consider lower order functions first. However in more dimensions it is not self-evident how to order the functions.
The following procedures have shown to lead to satisfactory results:

i) The maximum power of each variable is determined separately. This is done by assuming for a moment that the momentum function is dependent on that one variable alone. We thus make, for each variable in succession, a regular least squares fit in one dimension up to a power which produces no significant reduction in the sum of squares of residuals. If this power should be higher than seven, say, we would divide the range of that variable into sub-intervals as recommended in rules 5 and 10 at the beginning of this section.

ii) Having determined for each variable its maximum power, the index \( i \) is ordered for increasing values of

\[
Q = \frac{j}{J} + \frac{k}{K} + \frac{\ell}{L} + \frac{m}{M} + \frac{n}{N},
\]

where \( J \) is the maximum value of \( j \), \( K \) the maximum of \( k \), etc. (see Fig. 11).

The maximum value of \( i \) is then put equal to the one corresponding to, for example, \( Q = 1 \) or \( Q = 2 \) (rather than \( Q = 5 \)).

<table>
<thead>
<tr>
<th>( \xi_1 )</th>
<th>( \xi_1^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>( \xi_2 )</td>
<td>2</td>
</tr>
<tr>
<td>( \xi_2^2 )</td>
<td>3</td>
</tr>
<tr>
<td>( \xi_2^3 )</td>
<td>5</td>
</tr>
<tr>
<td>( \xi_2^4 )</td>
<td>7</td>
</tr>
<tr>
<td>( \xi_2^5 )</td>
<td>10</td>
</tr>
</tbody>
</table>

**Fig. 11** An example illustrating how the functions \( \xi_1, \xi_2 \) can be ordered. The heavy line indicates the limit for \( (i/I + j/J) \leq 1 \).
iii) Depending on whether each index is odd or even we can separate the model into \(2^5 = 32\) symmetry classes, in each of which the above two procedures can be applied. For procedure (i) the indices of the variables not under consideration are then put equal to 0 or 1. A difficulty here is the number of points. If we have a few hundred points \(x_\alpha\) in one dimension, we can safely assume that they are more or less symmetrically distributed, i.e.

\[
\sum_\alpha x_{2i+1}^\alpha \ll \sum_\alpha x_{2j}^\alpha
\]

for all integer \(i\) and \(j\) and \(-1 \leq x_\alpha \leq 1\).

If, however, these points lie in five dimensions there are only a few points in each of the 32 "quadrants". To assume that these are distributed the same way in each quadrant is asking too much. On the other hand, we only need this assumption to select those functions which are to be present in the final expression; the actual fit can later be made with all symmetries included.

iv) For each function under consideration we test whether its contribution (to the reduction of the sum of squares of residuals) is sufficiently big. To do this, using a non-orthogonal model, we need to take into account all functions already accepted in the model.

There is a way of doing this without actually making the fit (i.e. without matrix inversion). The way to do this is to first examine what part of the function under consideration is new (i.e. orthogonal). To clarify this, we consider the function values \(f_i(\xi_j)\) as a vector \(\vec{f}_i\) in \(N\)-dimensional space (\(N\) is the number of reference points). The known values of the momentum function \(P(\xi_j)\) thus form the "data vector" \(\vec{P}\). We can now define a set of orthogonal vectors, the \(i^{th}\) of which, \(\vec{w}_i\), is obtained from \(\vec{f}_i\) by successive subtraction of the projections on the \(i - 1\) orthogonal vectors \(\vec{w}_k\), where \(k = 1, 2 \ldots i - 1\)

\[
\vec{w}_i^{(0)} = \vec{f}_i \quad \quad (\vec{w}_1 = \vec{f}_1)
\]

\[
\vec{w}_i^{(k)} = \vec{w}_i^{(k-1)} - \left(\vec{w}_i^{(k-1)} \cdot \vec{w}_k\right) \vec{w}_k
\]

\[
\vec{w}_i^{(i-1)} = \vec{w}_i^{(i-1)}.
\]
The superscript indicates how many projections have been made. This is the so-called (modified) Gram-Schmidt orthogonalizing process\textsuperscript{16-18}). We thus construct an orthogonal model where the potential reduction in the sum of squares of residuals, $\Delta S_i$, can be readily evaluated for various choices of $\hat{f}_i$. This reduction is in fact given by the square of the projection of $\hat{f}_i$ onto the data vector:

$$\Delta S_i = (\hat{f}_i \cdot \vec{w}_i)^2/\vec{w}_i^2.$$  \hspace{1cm} (15)

The above four procedures reduce our multi-dimensional fitting problem sufficiently to be manageable. The first procedure typically reduces the total number of functions to a few thousands and the second reduces this again by an order of magnitude. Even if we do not separate the symmetries (because we may have an insufficient number of points) we can with the last procedure readily select up to a hundred of the most important functions. Having made already the orthogonal transformation the fit itself has become trivial (see Appendix 4).

The actual sum of squares of residuals can then be compared with the accumulated contributions given by Eq. (15); a difference must be due to taking a power in one of the variables which is sufficiently high for the machine rounding errors to become relevant.

Let us assume that we now have an explicit expression of the momentum $P$ in terms of the new variables (see Fig. 12 for a simple example). To test this we have to simulate new tracks. We typically find an r.m.s. deviation up to twice as high as the one on the reference tracks. This is due to the original sample being too small (a few hundred tracks). Occasionally we might find, however, a new track which is several standard deviations off. This is then probably due to the corresponding point in $M$-dimensional space lying outside the region covered by the reference points, so that we are in fact extrapolating rather than interpolating. I must, therefore, once more stress the importance of taking a sufficient number of reference points near or at the boundary of the region of interest.
Fig. 12  An actual example of the explicit expression of $P(\xi)$
($\xi$ is denoted by $x$ in the above program)

A tricky problem is the optimum number of terms in the final expression. The computations are subject to errors of measurement of the magnetic field. This, however, is only one of the reasons to limit the number of terms in the final expression. One must not "overfit", i.e. accept too many functions. In that case the sum of squares of residuals on the reference points will go down, but for new tracks the fit becomes worse. The easiest procedure is to fix beforehand (haphazardly or by successive trials) an upper limit to the total number, $I$, of functions (coefficients) and then only accept the $(i + 1)^{th}$ function if its contribution [see Eq. (15)] is sufficiently big, i.e.

$$\Delta S_{i+1} > \frac{S_1}{I_i - I_i},$$

where $S_1$ is the current sum of squares of residuals (see Fig. 13).
To shorten the final expression of $\vec{P}$ in terms of $\vec{\xi}$, we can, once more, use *a priori* knowledge. We can, for instance, as a simple approximation assume a homogeneous magnetic field. From the known detector positions and the dimensions of the approximated field we can then derive from $\vec{x}$ or from $\vec{\xi}$ an approximation $P^*$ of the quantity $P$. The multidimensional fit in terms of $\vec{\xi}$ is then made for the unexplained difference $P - P^*$ or the unexplained ratio $P/P^*$.

Up until here we have assumed that there is only one quantity, $P$, we are interested in. However, we saw that there may in fact be five such quantities. Let us now indicate these by a five-dimensional vector $\vec{P}$. Let us further assume that each component of $\vec{P}$ is already reduced by simple approximations using *a priori* knowledge. There is then still a further reduction possible of the, now five, final expressions $\vec{P}(\vec{\xi})$, the reason being that the components of $\vec{P}$ may be correlated. We therefore define five quantities.
\[ \vec{\xi}_i = \sum_j v_{ij} \vec{p}_j \]

to remove this correlation by a linear transformation similarly to Eq. (11). If there are only, say, three components of \( \vec{p} \) we are interested in, we only define three components of \( \vec{\xi} \).

3. SUMMARY

Let us now conclude by a brief summary of the method of using tabulated trajectories (see Fig. 14).

To evaluate the coefficients:

1) Define five quantities \( \vec{p} \) which define a track (e.g. two angles, the absolute value of the momentum, and the two space coordinates where the track passes a fixed plane).

2) Compute, for \( N \) different values of \( \vec{p} \), the tracks through the magnetic field and determine the corresponding \( M \) observations \( x_{i\alpha} \), where \( i = 1, \ldots, M \) and \( \alpha = 1, \ldots, N \).

Fig. 14a  Determining the coefficients  Fig. 14b  Processing actual observation
3) Use these simulated observations to determine with a simple approximation (e.g. assuming a homogeneous field) the values of $\mathbf{P}$. We call the values thus obtained $\mathbf{P}^*$. 

4a) Determine from $\mathbf{x}$ a set of at least five relevant coordinates $\mathbf{z}$ using a priori knowledge (e.g. two components to define a straight line in the vertical plane and three for defining a spline in the horizontal plane).

or alternatively

4b) Determine the $M$ eigenvectors $\mathbf{w}_l$ ($l = 1, \ldots, M$) of the correlation matrix $A_{ij} = \sum_\alpha (x_{i\alpha} - \bar{x}_i)(x_{j\alpha} - \bar{x}_j)$. Let only $L$ eigenvalues be significantly different from zero, then define

$$\zeta_j = \sum_{i=1}^{M} w_{ij} x_i \quad j = 1, \ldots, L,$$

where $w_{ij}$ is the $i^{th}$ component of the eigenvector corresponding to the $j^{th}$ largest eigenvalue.

Determine $M - 5$ constraint equations, $M - L$ of which are homogeneous.

5) Determine the five eigenvectors $\mathbf{v}_j$ of the correlation matrix

$$B_{ij} = \sum_\alpha \left( Q_{i\alpha} - \bar{Q}_i \right) \left( Q_{j\alpha} - \bar{Q}_j \right), \quad i = 1, \ldots, 5 \quad j = 1, \ldots, 5$$

where

$$Q_{i\alpha} = P_{i\alpha}/P^*_{i\alpha}$$

and define

$$\zeta_j = \sum_i v_{ij} Q_i \quad j = 1, \ldots, 5.$$ 

6) For each component of $\mathbf{z}$ make an $L$-dimensional fit, using $\mathbf{z}$ as variables, thus determining five sets of coefficients $\zeta_j$. 

Then to process actual observations we take the following steps:

1) Test whether a set of observations \( \mathbf{X} \) is real by using the constraint equations.

2) Determine \( \hat{P}^* \) by the same model as before.

3) Determine \( \hat{\xi} \) by the same transformation as before.

4) Use the coefficients \( \hat{\xi}_j \) to determine \( \hat{\xi} \).

5) Determine

\[
P_i = p_i^* \sum_j v_{ij} \xi_j.
\]

**Acknowledgements**

The author is greatly indebted to Mr. W. Eadie who drew his attention to the statistical formulae and furthermore to Dr. H. Øverås, Dr. M. Hine, Professor B. Zacharov, Professor Th. W. Ruijgrok, Miss J. Adams, Mr. D. Townsend, and several persons who furnished data, for their valuable contributions to discussions.
AN EXAMPLE OF INTRODUCING NEW VARIABLES BY MAKING A SPLINE FIT

We consider as an example the simple case that we have two detectors before a spectrometer magnet, two inside the field, and again two on the other side of the magnet; all six detectors are assumed parallel. For simplicity we consider the track of a charged particle in the horizontal projection only (see Fig. 3).

Even though the field is not assumed to be homogeneous, we can define an effective interval \([P, Q]\) over which the field is supposedly acting. The fit we want to make is that of two straight lines outside the field region and a parabola for \(P < x < Q\). We further impose continuity of the fitted function (track) and its first derivative in \(P\) and \(Q\).

We thus have as a model
\[
\begin{align*}
    y &= a_1 + a_2 x & \text{if} & \quad x < P \\
    y &= a_3 + a_4 x + a_5 x^2 & \text{if} & \quad P < x < Q \\
    y &= a_6 + a_7 x & \text{if} & \quad Q < x .
\end{align*}
\]

The continuity constraints are
\[
\begin{align*}
    a_1 + a_2 P &= a_3 + a_4 P + a_5 P^2 \\
    a_3 + a_4 Q + a_5 Q^2 &= a_6 + a_7 Q \\
    a_2 &= a_4 + 2a_5 P \\
    a_4 + 2a_5 Q &= a_7 . & \quad (A1.1)
\end{align*}
\]

We have the least squares criterion
\[
\sum_{\alpha=1}^{6} \left[ y_\alpha - \sum_i a_i f_i(x_\alpha) \right]^2 \text{ minimum} , \quad (A1.2)
\]

where
\[ f_i(x) = 1 \quad \text{for} \quad i = 1, 3, \text{and} \ 6, \]
\[ f_i(x) = x \quad \text{for} \quad i = 2, 4, \text{and} \ 7, \]
\[ f_5(x) = x^2 \quad \text{if} \quad P < x < Q, \]

and

\[ f_5(x) = 0 \quad \text{otherwise}. \]

We thus have seven linear equations obtained by differentiating formula (A1.2) with respect to \( a_i \) and four constraints formulae (A1.1). A solution can now be found by inverting an \( 11 \times 11 \) matrix. This is indeed done in the program SPLINT which was written by W. Eadie.

A good principle is, however, to build the constraints in the model; in this case we invert a \( 3 \times 3 \) matrix (7 - 4 instead of 7 + 4 equations).

In the above example we thus write the model as

\[
y = a_1 + a_2 x + a_3 (2Px - P^2) \quad \text{if} \quad x < P
\]
\[
y = a_1 + a_2 x + a_3 x^2 \quad \text{if} \quad P < x < Q
\]
\[
y = a_1 + a_2 x + a_3 (2Qx - Q^2) \quad \text{if} \quad Q < x.
\]

We have eliminated \( a_1, a_2, a_5, \) and \( a_7 \) from formula (A1.1) and re-named \( a_3, a_4, \) and \( a_5, \) as \( a_1, a_2, \) and \( a_3. \) After redefining \( f_i(x) \) the differentiation of formula (A1.2) now leads to three linear equations. Indeed we found results that are identical to those found with the SPLINT program.

Writing formula (A1.2) in vector notation we get

\[
\left( Y - FA \right)^2 \quad \text{minimum}
\]

\[
\frac{\partial}{\partial F} Y = \frac{\partial}{\partial F} Y
\]

\[
\hat{A} = \left( \frac{\partial}{\partial F} Y \right)^{-1} \frac{\partial}{\partial F} Y
\]

We can consider \( \hat{A} \) as new variables \( \xi \), and \( \left( \frac{\partial}{\partial F} \right)^{-1} \) is then the linear transformation \( w_{ij} \). If the coordinates observed, or generated by Monte Carlo computation, would indeed lie on a "spline curve" the value of \( \left( \frac{\partial}{\partial F} \right)^{-1} \) can be found by considering the coordinates only, ignoring any a priori knowledge.
APPENDIX 2

THE "HEDGEHOG MODEL" OF A MAGNETIC FIELD

The model for one magnetic field component\(^{12}\), which was used so far has the following properties:

1) Zero Laplacian.

2) Separable into eight symmetry classes and into functions of the three Cartesian coordinates.

3) Separable into a minor part, which does contribute to values at the twelve boundary edges of a rectangular box on the one hand, and the main part, which is zero at these edges on the other hand. This latter part of the model is orthogonal for equidistant observations on the boundary of a rectangular box.

We now present a model of the complete magnetic field, i.e. all three components, which has the following properties:

1) Zero divergence and zero rotation.

2) Separable into eight symmetry classes and into functions of the three Cartesian coordinates.

3) Separable into a minor part which can have non-zero slope at the above boundary edges and the main part which has zero slope at these edges\(^*\). This latter part is orthogonal for equidistant boundary observations of the component perpendicular to the boundary of a rectangular box.

This model is suited to make a fit to observations of

\[ B_z \text{ at } z = \pm Z_0/2 \text{ and at equidistant } x \text{ and } y, \]
\[ B_x \text{ at } x = \pm X_0/2 \text{ and at equidistant } y \text{ and } z, \]          \hspace{1cm} (A2.1)
\[ B_y \text{ at } y = \pm Y_0/2 \text{ and at equidistant } z \text{ and } x, \]

where \(-X_0/2 \leq x \leq X_0/2\) and similarly for \(y\) and \(z\).

(The notation is slightly different from the one used in Ref. 12.)

\(^*\) For a variation perpendicular to the edge.
The Model

Let \((x, y, z), (X_0, Y_0, Z_0), (i, j, k), (\ell, m, n), (a_{ij}, b_{jk}, c_{ki})\) be given. The formulae below are also valid when the above elements in brackets are rotated simultaneously.

We define

\[
\begin{align*}
    i^* & \equiv (3 - \ell) \frac{X}{X_0} \\
    j^* & \equiv (2i - 2 - \ell) \frac{Y}{Y_0} \\
    R_{ij} & \equiv (i^* + j^*) \frac{1}{2} Z_0 \\
    f_{i\ell}(x) & \equiv \sin (i^* x) \\
    f_{i\ell}(x) & \equiv \cos (i^* x) \\
    g_{ij} n(z) & \equiv Z_0 \sinh (R_{ij} z / Z_0) / \left[ R_{ij} \cosh (R_{ij} / 2) \right] \\
    g_{ij} n(z) & \equiv Z_0 \cosh (R_{ij} z / Z_0) / \left[ R_{ij} \sinh (R_{ij} / 2) \right]
\end{align*}
\]

except for \(\ell = m = 2\), where we define

\[
\begin{align*}
    g_{221}(z) & \equiv 0 \\
    g_{222}(z) & \equiv 0 \\
    g_{221}'(z) & \equiv 1 \\
    g_{222}'(z) & \equiv 2z / Z_0.
\end{align*}
\]

The intelligent reader will notice that although \(g_{ij} n(z)\) does not have the indices \(\ell\) and \(m\), it still depends on them via \(R_{ij}, i^*\) and \(j^*\).

\[
\mathbf{B}_{\ell mn} \equiv \mathbf{\nabla} \cdot \psi_{\ell mn} \quad \text{and therefore} \quad \mathbf{\nabla} \times \mathbf{B}_{\ell mn} = 0.
\]

We note

\[
\begin{align*}
    f''_{i\ell}(x) & = -i^* x \frac{1}{2} f_{i\ell}(x) \\
    g''_{ij} n(z) & = (i^* + j^*) g_{ij} n(z)
\end{align*}
\]

consequently \(\Delta \psi_{\ell mn} = 0\), and therefore \(\mathbf{\nabla} \cdot \mathbf{B}_{\ell mn} = 0\). The ' stands for differentiation.
The definition (A2.2) of $g_{222}(z) \equiv 0$ would not seem logical. However, this function is only used together with a factor such as $i^\ast \sin (i^\ast x)$, with $i^\ast = 0$. We choose the above definitions to be able to evaluate the coefficients $a_{ij}$ from the $B_z$, $b_{jk}$ from $B_x$, and $c_{ki}$ from the $B_y$ observations independently. However, to justify this we have to add the one constraint that for $\ell = m = n = 2$,

$$a_{22} + b_{22} + c_{22} = 0.$$ 

**Evaluating the coefficients**

To evaluate the coefficients $a_{ij}$, $b_{jk}$, and $c_{ki}$ to fit observations as given in Eqs. (A2.1), we first define new data that are separated into eight symmetry classes by taking sums and differences. It should be noted that these symmetries refer to $\psi$ so that, for example, a three-fold symmetric $B_z$ field would be labelled $\ell = 2$, $m = 2$, $n = 1$, as a result of the differentiation of $\psi$.

For each symmetry class we then proceed as follows:

1) Make a fit to the $B_z$ observations evaluating $a_{ij}$. Except for $a_{11}$ and $a_{1j}$, $a_{ij}$ are coefficients of an orthogonal model. To get $a_{1j}$ one needs estimates of $dB_z/dx$ at $x = X_0/2$, $z = Z_0/2$, and similarly for $a_{11}$. It may be helpful to use the equalities

$$\frac{dB_z}{dx} = \frac{dB_x}{dz} \quad \text{and} \quad \frac{dB_z}{dy} = \frac{dB_y}{dz}.$$ 

2) Subtract from the $B_x$ observations

$$\sum_j a_{1j} f'_1(X_0/2) f_{jm}(y) g_{1jm}(z).$$

3) Make a fit to the residuals of $B_x$ evaluating $b_{jk}$. To get $b_{1k}$ one needs estimates of $dB_x/dy$ (= $dB_y/dx$) at $x = X_0/2$, $y = Y_0/2$.

4) Subtract from the $B_y$ observations

$$\sum_i a_{1i} f'_1(x) f'_{1m}(Y_0/2) g_{1in}(z) + \sum_j b_{1j} f'_{jm}(Y_0/2) f_{1n}(z) g_{1j}(x).$$
5) Make a fit to the residuals of the \( B_y \) observations evaluating \( c_{ki} \).

6) For \( \lambda = m = n = 2 \) define

\[
\delta = (a_{22} + b_{22} + c_{22})/3
\]

and redefine

\[
\hat{a}_{22} = a_{22} - \delta, \quad \hat{b}_{22} = b_{22} - \delta \quad \text{and} \quad \hat{c}_{22} = c_{22} - \delta.
\]

When not all three components are measured, the above steps 1-6 are to be reduced to 1-4 (for \( B_z \) and \( B_x \) observations only) or just 1 for \( B_z \) observations only. In the first case \( c_{ki} \) and in the second case \( b_{jk} \) and \( c_{ki} \) are put equal to zero.
APPENDIX 3

HOW TO FIT AN L-DIMENSIONAL HYPERPLANE THROUGH N POINTS IN M DIMENSIONS

The transformation

We consider a case of N points each with M coordinates

\[ x_{\alpha j}, \quad \alpha = 1, \ldots, N, \quad j = 1, \ldots, M. \]

The first operation is to choose the origin at the centre of gravity, i.e. \( x_{\alpha j} \) is replaced by \( x_{\alpha j} - (1/N) \sum_{\alpha=1}^{N} x_{\alpha j} \) for each j. From now onwards we will assume that \( \sum_{\alpha=1}^{N} x_{\alpha j} = 0 \). We further assume that the random error of measurement, if any, has the same r.m.s. deviation for each j and each \( \alpha \). If it is not the same for each j then we scale \( x_j \) accordingly.

If some of the coordinates are correlated, then we can think of the points as having a sort of ellipsoidal distribution. The major axis of this ellipsoid is the best straight line we can fit through these points. But let us be more precise. Let \( \hat{w} \) be the direction cosines of a line through the origin; \( \hat{w}^2 = 1 \). We now use vector notation in M-dimensional space.

The best fit is obtained when the sum of squares of the distances to this line is a minimum, i.e.

\[ \sum_{\alpha} \left[ \hat{x}_{\alpha}^2 - (\hat{x}_{\alpha} \cdot \hat{w})^2 \right] \text{ minimum.} \quad (A3.1) \]

If we write

\[ \sum_{\alpha} \left[ \hat{x}_{\alpha}^2 - (\hat{x}_{\alpha} \cdot \hat{w})^2 \right] = \sum_{\alpha} \hat{x}_{\alpha}^2 - \sum_{\alpha} \sum_{i} \sum_{j} \hat{x}_{\alpha i} \hat{x}_{\alpha j} \hat{w}_i \hat{w}_j, \]

then, with the normalization constraint

\[ \hat{w}^2 = 1, \]

we find by differentiation with respect to \( \hat{w}_i \)

\[ 2 \sum_{\alpha} \sum_{j} \hat{x}_{\alpha i} \hat{x}_{\alpha j} \hat{w}_j = 2\lambda \hat{w}_i, \]

where \( \lambda \) is a Lagrange multiplier.
Writing this equation as
\[ \hat{A} \hat{w} = \lambda \hat{w}, \]
where
\[ A_{ij} = \sum_{\alpha} x_{\alpha i} x_{\alpha j}, \]
we see that the extrema of (A3.1) are given by the eigenvectors of \( \hat{A} \), the "dispersion matrix". The minimum value of (A3.1) is then
\[ \sum_{\alpha} x_{\alpha}^2 (\hat{A} \hat{w}) \cdot \hat{w} = \sum_{\alpha} x_{\alpha}^2 - \lambda \hat{w}^2 = \sum_{\alpha} x_{\alpha}^2 - \lambda, \]
where \( \lambda \) is the eigenvalue belonging to the above eigenvector. The best line is consequently the eigenvector belonging to the largest eigenvalue \( \hat{A} \) being positive definite, all eigenvalues are non-negative).

We now look for the best plane fitted, and we note that the projections of the residuals on the eigenvector belonging to the largest but one eigenvalue are equal to the projections of the original vectors \( \hat{x}_\alpha \) on this eigenvector, because eigenvectors are orthogonal. The first two eigenvectors, therefore, define the best fit of a plane through \( \hat{x}_\alpha \).
Thus, for each new eigenvector, \( \hat{w}_\ell \) included, the sum of squares of residuals, i.e. the sum of squares of the distances to the fitted subspace, is lowered by \( \lambda_\ell \). The original sum of squares, being the same when expressed in the coordinate system defined by all \( M \) eigenvectors, is
\[ \sum_{\alpha} x_{\alpha}^2 = \sum_{\alpha} \sum_{\ell} (\hat{x} \cdot \hat{w}_\ell)^2 = \sum_{\ell} \lambda_\ell. \]

Consequently the original sum of squares is the sum of all eigenvalues, and the residual sum of squares is the sum of the eigenvalues not used. The eigenvalues therefore measure the "goodness" of the fit and the "relevance" of a new vector included.

We now define \( L \) new coordinates \( \xi_\ell \) as the component of the observation \( \hat{x} \) in the direction \( \hat{w}_\ell \):
\[ \xi_\ell = \sum_i w_{i\ell} x_i, \quad \ell = 1, \ldots, L, \quad L \leq M. \]
The second index of \( w \) labels the corresponding eigenvalue.

Since

\[
\lambda_l = \sum_{\alpha} \xi_{\alpha l}^2 ,
\]

we expect \( \xi_{\alpha l}^2 \) to decrease at the same rate as \( \lambda_l \) for increasing \( l \).

The reconstruction of the original observation \( x_i \) can now readily be done:

\[
x_i = \sum_{j}^{M} w_{ij} \xi_j ,
\]

the reason being that the inverse of an orthonormal matrix is its transpose.

If \( \xi_l \) is not significantly different from zero for \( l > L \) then

\[
x_i \approx \sum_{j}^{L} w_{ij} \xi_j .
\]

Fig. 15 Defining new variables can reduce the region of interest. For more dimensions this becomes more important.
When the number of variables cannot be reduced (i.e. \( L = M \)), performing the above transformation can still offer the advantage of reducing the range of interest (see Fig. 15).

A computer program for making the transformation

In order to avoid storing all \( N \times M \) coordinates (\( N \) points in \( M \) dimensions) in the FORTRAN written program, the dispersion matrix is "updated" with each new observation \( x_{i\alpha} \).

This is done by the following formulae:

\[
\begin{align*}
\overline{x_i^{(1)}} & = x_{i1} \\
\overline{x_i^{(n)}} & = \overline{x_i^{(n-1)}} + \frac{1}{n} \left( x_{in} - \overline{x_i^{(n-1)}} \right) \\
A_{ij}^{(1)} & = 0 \\
A_{ij}^{(n)} & = A_{ij}^{(n-1)} + \frac{1}{n-1} \left\{ \left( x_{in} - \overline{x_i^{(n)}} \right) \left( x_{jn} - \overline{x_j^{(n)}} \right) - \frac{n-1}{n} A_{ij}^{(n-1)} \right\}
\end{align*}
\]

It can readily be shown that for each \( n \) this results in

\[
A_{ij}^{(n)} = \frac{1}{n} \sum_{\alpha=1}^{n} (x_{i\alpha} - \overline{x_i}) \cdot (x_{j\alpha} - \overline{x_j}),
\]

where

\[
\overline{x_i} = \frac{1}{n} \sum_{\alpha=1}^{n} x_{i\alpha}.
\]

Writing the formulae in the above form, rather than, for example,

\[
A_{ij} = \frac{1}{n} \sum_{\alpha} x_{i\alpha} x_{j\alpha} - \left( \frac{1}{n} \sum_{\alpha} x_{i\alpha} \right) \left( \frac{1}{n} \sum_{\alpha} x_{j\alpha} \right),
\]

has the additional advantage of avoiding the accumulation of machine rounding errors.

The eigenvalues of \( \overline{X} \) are determined by the CDC MATRIX subroutine, using a "combination of Householder's method for tridiagonalizing and a Sturm sequence". The equations for the eigenvectors are then solved by matrix inversion for each eigenvalue separately.
An indication of the computer time is obtained from the following test runs on the CDC 6600:

- 4 sec for \( N = 2000, \ M = 15 \) and \( L = 5 \),
- 250 sec for \( N = 10000, \ M = 80 \) and \( L = 10 \),

where \( L \) eigenvectors are determined for \( N \) points in \( M \) dimensions. Not more than 25 k words storage space was needed for any of the above-mentioned jobs. The above times are upper limits; they include, for example, reconstruction and orthogonality tests.

Anyway this time is, in principle, only needed once per experimental set-up.
THE GRAM-SCHMIDT TRANSFORMATION

We consider an $M \times N$ matrix $\mathbf{F}$, an element of which is given by

$$ f_{ij} = F_j(\xi_{1i}, \xi_{2i}, \ldots, \xi_{Li}), $$

where $i$ labels the $M$ observations and $j$ labels $N$ functions of $L$ variables ($N \leq M$).

We now want to determine coefficients $c^j$ such that $(\mathbf{d} - \mathbf{F} \mathbf{c})^2$ is a minimum, where $d_i$ are the known function values, say the momentum at the point $(\xi_{1i}, \xi_{2i}, \ldots, \xi_{Li})$.

We make use of the Gram-Schmidt orthogonalizing process. By this is meant that we define a matrix $\mathbf{W}$ of which the columns $\mathbf{w}_j$ are given by:

$$ w_i = f_i \quad \{= F_i(\xi_{1i}, \xi_{2i}, \ldots, \xi_{Li}) \text{ with } i = 1, 2 \ldots M \} $$

$$ w_j = f_j - \sum_{k=1}^{j-1} (f_j \cdot w_k) w_k/w_k^2. \quad (A4.1) $$

Then $w_i \cdot w_j = 0$ if $i \neq j$ (see further).

We now take as a new model

$$ \mathbf{w} \rightarrow \mathbf{w}^* \mathbf{a}. $$

We thus want to minimize

$$ S = (\mathbf{d} - \mathbf{w}^* \mathbf{a})^2. $$

Differentiation with respect to $a_j$ gives (since $w_j \cdot w_i = 0$ if $i \neq j$)

$$ d \cdot w_j - a_j w_j^2 = 0 $$

or

$$ a_j = (d \cdot w_j)/w_j^2. \quad (A4.2) $$

To derive now $c$, we first note that formula (A4.1) can be written as

$$ \mathbf{F} = \mathbf{W} \mathbf{B}, $$
where
\[ b_{ij} = (\hat{\ell}_{ij} \cdot \hat{w}_i) / \hat{w}_i^2 \quad \text{if } i < j \]
\[ b_{ij} = 1 \quad \text{if } i = j \]
\[ b_{ij} = 0 \quad \text{if } i > j. \]

Consequently, $\hat{B}$ is an upper triangular matrix, which can readily be inverted (see further on).

We now evaluate
\[ F \hat{B}^{-1} = \hat{W}. \]

The model $\hat{W} \hat{a}$ can therefore be written as $(F \hat{B}^{-1}) \hat{a} = F(\hat{B}^{-1} \hat{a})$.

The original model $F \hat{c}$ is therefore identical with this if
\[ \hat{c} = (\hat{B}^{-1} \hat{a})' = \left[ \hat{a}' (\hat{B}^{-1})' \right]' . \]

The reason why we use $(\hat{B}^{-1})'$ rather than $\hat{B}^{-1}$ is to save storage, since $(\hat{B}^{-1})'$ can be stored in the same matrix as $\hat{B}$.

The more coefficients we evaluate, the more the sum of squares of residuals $S$ will be reduced. We can evaluate $S$ before inverting $\hat{B}$.

Let $S_j$ be the sum of squares of residuals when taking $j$ functions into account. Then
\[ S_j = \left( \hat{d} - \sum_{i=1}^{j} a_i \hat{w}_i \right)^2 = \hat{d}^2 - 2 \hat{d} \left( \sum_{i=1}^{j} a_i \hat{w}_i \right) + \sum_{i=1}^{j} a_i^2 \hat{w}_i^2 . \]

We derived [see formula (A4.2)]:
\[ (\hat{d} \cdot \hat{w}_i) = a_i \hat{w}_i^2 , \]
so that
\[ S_j = \hat{d}^2 - 2 \sum_{i=1}^{j} a_i \hat{w}_i + \sum_{i=1}^{j} a_i^2 \hat{w}_i^2 = \hat{d}^2 - \sum_{i=1}^{j} a_i^2 \hat{w}_i^2 . \]

So for each new function $f_j$ included in the model we get a reduction of the sums of squares of residuals of $a_j^2 \hat{w}_i^2$ [where $\hat{w}_i$ is given by formula (A4.1) and $a_j$ by (A4.2)]. This way we can decide if we want to include this function in the final model before the matrix inversion.
Orthogonality with Gram-Schmidt transformation

The transformation is
\[
\hat{\mathbf{w}}_1 = \hat{\mathbf{f}}_1
\]
\[
\hat{\mathbf{w}}_i = \hat{\mathbf{f}}_i - \sum_{k=1}^{i-1} \left( \hat{\mathbf{f}}_i \cdot \hat{\mathbf{w}}_k \right) \hat{\mathbf{w}}_k / \hat{\mathbf{w}}_k^2 .
\]

Consider the statement:
\[
\hat{\mathbf{w}}_i \cdot \hat{\mathbf{w}}_j = 0 \quad \text{for } j > 1 \text{ and } i < j
\]

This statement is true for \( j = 2 \):
\[
\hat{\mathbf{w}}_1 \cdot \hat{\mathbf{w}}_2 = \hat{\mathbf{w}}_1 \{ \hat{\mathbf{f}}_2 - (\hat{\mathbf{f}}_2 \cdot \hat{\mathbf{w}}_1) \hat{\mathbf{w}}_1 / \hat{\mathbf{w}}_1^2 \} = \hat{\mathbf{w}}_1 \cdot \hat{\mathbf{f}}_2 - \hat{\mathbf{f}}_2 \cdot \hat{\mathbf{w}}_1 = 0 .
\]

Assume that it is true for \( j = 2, 3, \ldots, \ell - 1 \), then
\[
\hat{\mathbf{w}}_1 \cdot \hat{\mathbf{w}}_\ell = \hat{\mathbf{w}}_1 \left[ \hat{\mathbf{f}}_\ell - \sum_{k=1}^{\ell-1} (\hat{\mathbf{f}}_\ell \cdot \hat{\mathbf{w}}_k) \hat{\mathbf{w}}_k / \hat{\mathbf{w}}_k^2 \right]
\]
\[
= \hat{\mathbf{w}}_1 \cdot \hat{\mathbf{f}}_\ell - \hat{\mathbf{f}}_\ell \cdot \hat{\mathbf{w}}_1 = 0 .
\]

Hence the statement is true for every \( j > 1 \), and since \( \hat{\mathbf{w}}_1 \cdot \hat{\mathbf{w}}_j = \hat{\mathbf{w}}_j \cdot \hat{\mathbf{w}}_i \) the orthogonality holds.

Inverse of a triangular matrix

Let \( \hat{\mathbf{B}} \) be a square matrix with
\[
b_{ij} = 0 \quad \text{if } i > j \quad \text{and} \quad b_{ij} = 1 \quad \text{if } i = j .
\]

Further, let \( \hat{\mathbf{A}} \) be a square matrix with
\[
a_{ij} = 0 \quad \text{if } i > j \quad \text{and} \quad a_{ij} = 1 \quad \text{if } i = j ,
\]
and
\[
a_{ij} = - \sum_{k=i+1}^{j} b_{ik} a_{kj} \quad \text{if } i < j ,
\]
then for $i < j$

$$\sum_{k=1}^{j} b_{ik} a_{kj} = \sum_{k=i}^{j} b_{ik} a_{kj} = b_{ii} a_{ij} + \sum_{k=i+1}^{j} b_{ik} a_{kj} = a_{ij} - a_{ij} = 0,$$

for $i = j$

$$\sum_{k=1}^{n} b_{ik} a_{kj} = b_{ii} a_{ii} = 1,$$

and for $i > j$

$$\sum_{k=1}^{n} b_{ik} a_{kj} = 0.$$

Therefore $A^{-1} A = I$ and since $A$ and $B$ are square and non-singular, $A = B^{-1}$.

The modified Gram-Schmidt transformation

Formula (A4.1) can also be written as

$$\vec{w}_j^{(0)} = \vec{f}_j$$

$$\vec{w}_j^{(k)} = \vec{w}_j^{(k-1)} - (\vec{w}_j^{(k-1)} \cdot \vec{w}_k^{(k-1)}) \vec{w}_k^{(k-1)} / \vec{w}_k^{2}$$

$$\vec{w}_j^{(j-1)} = \vec{w}_j^{(j-1)}.$$

It is interesting to note that the modified Gram-Schmidt transformation\(^{17}\) which leads to the same transformation, but is less subject to machine rounding errors\(^{18}\), can be written as

$$\vec{w}_j^{(0)} = \vec{f}_j$$

$$\vec{w}_j^{(k)} = \vec{w}_j^{(k-1)} - (\vec{w}_j^{(k-1)} \cdot \vec{w}_k^{(k-1)}) \vec{w}_k^{(k-1)} / \vec{w}_k^{2}$$

$$\vec{w}_j^{(j-1)} = \vec{w}_j^{(j-1)}.$$

This modification thus meant only one character change in the earlier FORTRAN program!
REFERENCES

1) W.T. Eadie, D. Dryard, F.E. James, M. Roos and B. Sadoulet, Statistical methods in experimental physics (North-Holland, Amsterdam, 1971), Section 8.4.4.


7) C. Lanczos, Applied analysis, p. 236.


9) B. Braams, Private communication, June 1972.


18) Å. Björck, BIT 7, 1 (1967).
SMALL COMPUTERS IN PHYSICS

I. C. Pyle

Atomic Energy Research Establishment,
Harwell, U.K.

1. GENERAL ORGANISATION OF SMALL COMPUTERS

1.1 Introduction

Over the last ten years, the role of the computer in experimental physics has changed from being a luxury toy to being an essential tool. This change happened somewhat earlier in theoretical physics, in which 'number crunching' computers have played a vital role since their earliest days. In experimental physics, the important computers have been the small ones, as opposed to the large ones for theoretical physics. This has been primarily because of two factors:

(a) their cost has been so low that projects have been able to afford them on a dedicated basis - a piece of major experimental apparatus can have its own associated computer, and

(b) they have had the technical capability, in both hardware, software and processing speed, to be attached on-line to their apparatus servicing it in real time.

1.2 Characteristics of small computers

Before pursuing these points, let us note the characteristics of a (digital) computer:

SLIDE 1.01

The last characteristic is the one which technically underlies the ability of the computer to be used on-line. There is no clear-cut definition of a small computer, only a set of parameters which are particularly relevant in discriminating the 'small' from the 'large'.

SLIDE 1.02

It will be observed that some of these criteria also apply to modern large computers. This is well worth noting, because notwithstanding the claims of some computer manufacturers that the prestige and the
technical advances came from the largest computers, there are many important respects in which small computers have led the way. Indeed, for the largest computing systems, it is better to get designers with a background in small computers (who are used to having to live with hardware constraints) rather than those used to large computers (who approach problems with the expectation that their computer will be more than powerful enough to do their job and someone else's too at the same time).

In the various studies that there have been of small computers, it has been common to classify them into three groups, on the basis of word length, adaptability (i.e. types of peripherals or other equipment which can be attached) and application complexity (i.e. number of concurrent tasks to be done). The price ranges for basic configurations of computers of these types overlap, but can be distinguished.

SLIDE 1.03

At the Mini and Midi levels, the adaptability extends to being able to handle peripherals which the computer manufacturer never dreamed of (e.g. an X-ray diffractometer). The physics laboratory consequently has to face the problems of interfacing, and making control devices for their apparatus which can interact with the computer. Small computer manufacturers on the whole recognise that these problems exist, and aim to make life tolerable for the people trying to solve them; large computer manufacturers tend either to deny that the problems exist or to fail to do anything about them.

A traditional dichotomy in the applications of computers has been between the scientific and the commercial: numerical processing against organisational processing, slight input/output against bulky input/output. The small computer in physics does not come into this spectrum, but sets up a new dimension.

SLIDE 1.04

Traditional computing has the ultimate-user input/output via an off-line device (e.g. card key-punch) and a response time measured in hours or at best tens of minutes. In contrast, the application for a typical small computer (which I call 'immediate' processing)
involves ultimate input/output on-line (connections to the apparatus, or to an operator display) and a response time measured in seconds or milliseconds. It also involves data processing of a structural nature, not predominantly numerical, and there may be a considerable volume of data to be handled on the fly. If a further subclassification of 'immediate' processing is required, one can distinguish

real time: apparatus on-line: response is in milliseconds;
interactive: human user on-line: response is in seconds.

This course is concerned with real-time and interactive use of small computers, particularly mini and midi computers.

Although many of the remarks are expressed in general terms, I try to make the examples specific, so that particular points are brought out realistically. Small computers are genuine engineering and commercial compromises, and it is important to let particular constraints show. My main examples are taken from the PDP-11 of DEC, but I also refer to the Honeywell DDP 516 and the Modular One of Computer Technology Ltd. One or two others are mentioned in passing e.g. the PDP-8.

1.3 Hardware Architecture

The hardware architecture of the computers is primarily characterised by the way in which the processor, store modules, and input/output controllers are connected together.

SLIDE 1.05

Connections are needed at least between the processor and the other two; high speed data transfers require connections between the input/output controllers and the store.

There are two different architectural parameters which apply: the nature of the processor - i/o controller link may be the same or different from that of the processor - store link; and the distribution of links between the modules may be as a net or as a bus.

The DDP 516 has quite different i/o connections from its store connections: this is reflected in its order code containing explicit i/o instructions. Similarly the PDP-8 has a distinct i/o mechanism. On the other hand, the Modular One and the PDP-11 processors both
treat their i/o controllers very much like store modules. They have no special i/o instructions, but simply use the top 4k words of addressing space to reference peripheral controllers. (There is a minor difference between store and i/o controller connections for a Modular One processor, and separate ports are provided on the processor. This does not show on the FDP-11.)

The bus-web issue is an economic compromise between equipment and data throughput. The web architecture of the Modular One permits independent data transfers to be going on over difficult links genuinely at the same time, but at a cost which mounts up alarmingly as you develop a system. Conversely, the bus architecture of the FDP-11 is economical in equipment, but because all data transfers go through the same channel, contention can arise, and the system can be put at risk by excessive use of the bus by rogue equipment. (This can arise with certain CAMAC couplers.)

1.4 Processors, stores and controllers

As well as their active components for performing instructions, processors contain some internal memory. This is important, for it is the contents of the internal memory which distinguishes and gives the complete context for the particular task the processor is doing. A physically identical processor doing a different task would just have different states in its internal memory. We refer to these collectively as its 'live registers', and note that to change the contents of the live registers of a processor amounts to changing the task it is doing. It does not usually make sense to change some of these and not others - the whole lot belongs together. The live registers are mainly concerned with control: address of next instruction to be executed, condition codes, interrupt mask; they also usually include some data registers (accumulators, index registers). The processor not only executes a sequence of instructions following its next instruction register, but also is sensitive to exceptions and external stimuli which may come to it from its i/o controllers (remember the last characteristic on slide 1.01). Apart from periods when interrupts are inhibited, the processor senses the interrupt request status before extracting a new instruction to
execute, and takes an alternative path if an interrupt is requested. Different processors have slightly different sets of live registers, and different ways of responding to an interrupt request.

Store modules typically contain 1K, 4K or 16K words, and are almost universally core store, at speeds often going down to 750 ns; in some cases semiconductor memory modules can be used to get even faster speeds. Semiconductor memory loses its information on a power failure, so if you have semiconductor memory you usually have another medium to back it up.

Input/output controllers act between the processor and the peripheral input/output devices themselves, which are either data transducers or backing storage devices. Often several devices of the same type can be handled by the same controller. Controllers differ markedly in their power, depending on the complexity of the task they have to do and the speed of their subject device. Here is another area of compromise between putting equipment into the controller to save processor time, and having a simple controller which requires more software and processor time to make it work.

1.5 Interrupt hardware and interrupt handling

The emphasis on input/output, especially with foreign attachments to the computer, which we have seen, makes it essential that the computer responds rapidly when an interrupt request arises, and that the program which determines the response should be flexible. We can distinguish two problems here:

(a) when to make the processor do some necessary work
(b) what processing on the data is required.

In this lecture, we will concentrate on the first of those, leaving the second until later.

Fundamentally, the i/o device asserts when it is ready (having gathered some data for input, or disposed of data it was given to output); the processor has to detect this and respond appropriately. The processor must look at the signal from the i/o device sufficiently often to be able to respond within a certain time specific to the device, otherwise some sort of disaster will happen. This
time is called the crisis time, or patience, of the device. (see Middleton 1972).

SLIDE 1.06

Testing whether an i/o device is ready can be done either by software (polling) or hardware (interrupt). For polling, the controller simply sets a bit (which we call a flag) in a register which the program can test. The method relies on the program running at the time containing the necessary test instruction, and making sure that it gets executed at appropriate intervals. If the test succeeds, then the appropriate responding program is entered. But if the running program fails to make the test (e.g. gets into a loop not containing a test), then the opportunity to respond is lost, and the whole input/output operation is damaged.

For the interrupt method, the controller sets a bit in the same way, but this is brought directly into the processor, which tests it automatically (by hardware) between consecutive instruction executions. In this case, there is no problem about sampling the status: the problem is rather of controlling the responding program to prevent it from garbling the background program which was previously running, by properly exchanging the live registers of the processor.

Polling expressly permits interruptions at particular points (break-points) in the program. The background program has the necessary ability and the responsibility, to take care that it only does the test at places where the live registers contain standard or non-significant values. Interrupting on the other hand requires an inhibition mechanism to prevent the automatic interrupt taking place during critical sections of the running program, during which an interrupt if permitted would garble the context. A program must be able to turn to the inhibition on and off.

The most important case of this is at the extrema of a responding process. These extrema, concerned more with the interrupt mechanism than with the response to be computed, are conveniently handled by a general purpose piece of software, which is called the interrupt handler. The beginning is concerned with saving the live registers for the background process, before starting on the body
of the responding process; the end restores those live registers to resume the background process. During each of these save/restore sequences, no further interrupts must occur. Since there are usually several live registers, needing several instructions to save/restore them, interrupts must be prevented from being effective during those sequences.

SLIDE 1.07

Therefore, when an interrupt happens, as well as starting the interrupt handler for the responding process, the processor must be inhibited from accepting further interruptions until the responding process expressly permits. The final stage of the interrupt handler similarly needs interrupts off, which is all right accept for the final instruction to permit further interrupts - this must be followed by an instruction to transfer control back to the background process, and interrupts must not be allowed to come between these. Hence a 'permit interrupts' instruction must cause the interrupts inhibit mechanism to come off after the following instruction has been executed.

1.6 Input/Output Organisation

Input/Output controllers may operate on different sized blocks of data depending on the speed of the device, usually single bytes or words of data (slow devices) or device dependent blocks of data (fast devices).

SLIDE 1.08

Even with the simplest devices, the controller has to contain some registers for holding control and status information. For controllers handling several similar devices, a form of multiplexing is involved, and a select register has to specify which of the devices is the subject of the current transfer. Different sorts of device need different sets of signals for control, and produce different status signals.

Because it is desirable that user programs be as far as possible device independent, these inevitable differences call for an intermediate layer of software with appropriately specific interfaces to the device controllers, and a uniform device-independent inter-
are the basic and traditional languages used for programming small computers, with one line per instruction and additional statements for data space assignment, data value specification, inter-module link naming, and assembly control. The new idea of intermediate level languages, was introduced by Wirth (1968) in PL/360

SLIDE 2.04

and widely followed in for example PL/516 (Bell & Wichmann, 1971) and PL/11 (Russell, 1971). We express instructions not in individual statements but as (still explicit) parts of a structured statement. Thus the programmer still has control over the instructions generated, but the source language is more concise and comprehensible.

The high level languages translated by a compiler can be more highly structured, and there can be a less direct relationship between the source and object code due to optimisation. Examples are Coral 66 (Woodward, Weatherall and Gorman, 1970)

SLIDE 2.06

and LSD (Calderbank and Calderbank, 1971).

SLIDE 2.07

The hardware of the computer establishes certain conventions for software: instruction format, address modification, integer data representation, floating point data representation, etc. At the assembler level, these are the only conventions established firmly, and the programmer is in principle free to make his own conventions for other items: subroutine linkage, array layout, string data representation etc.

SLIDE 2.08

As we move up the spectrum, we find that additional conventions become established by the software, and these handle in standard ways some of the programmer's problems.
2.4 Library Subroutines

The linker can join together the loose ends of a set of modules presented to it, and then resolve the remaining unsatisfied references by extracting appropriate modules from a library. The library subroutines provide standard ways of doing many things - not just mathematical functions but also file access and much data manipulation. The library subroutines may imply new software conventions for data representation. They are frequently necessitated by higher-level languages, in which common facilities can often be implemented more economically by an external subroutine rather than by expansion of code in-line, since only one copy of the subroutine is then needed for the whole collection of subroutines linked together. Particular examples are the set of subroutines needed to implement Fortran input/output statements, and all the device handlers.

The effect of having a library is to enhance the facilities of the bare machine, in a selective way. Store is occupied by a module only if it is referenced (directly or indirectly) by the user's program. Facilities which are not referenced do not take up store for this user's program, although they are available to any user who wants them. You don't pay (in store) for what you don't reference!

This is not quite the same as dynamic linkage, in which store is occupied by a module only if it is actually called for during execution of the user's program. This is a balance of store against time. To get the module loaded when called for would take up more time during execution, introducing greater complication at run-time, and require a dynamic linker which would take up more space than had been saved.

2.5 Stacks and Queues

Particular data structures which are needed in system programming are certain forms of list structure, which are less well known than arrays.

Whereas an array, having been created with a certain number of
elements, always has the same number of elements, these list structures are variable in size. When created, they have to be allocated storage which implies a specific maximum size, but apart from that they can dynamically contain any number of elements from zero upwards. Important operations are consequently not so much 'read' and 'write' as 'put in' and 'take out'. Lists which can be accessed only at their ends can be stored compactly and efficiently, and the two important forms of this type are stacks and queues.

SLIDE 2.09

A stack has one end of the list fixed and the other end free: 'put in' and 'take out' operations both apply at the free end. With a stack, 'put in' is called 'push', and 'take out' is called 'pop'. This implements a nesting situation, in which a matched sequence of inner activities can be performed without affecting outer activities on a longer time scale.

A queue has both ends of the list mobile, with one end used only for 'put in' operations and the other end used for 'take out' operations. This implements a buffering situation, in which the relative order of events is maintained, but absolute times are not.

A list which can be accessed in the middle as well as at the ends is rather more cumbersome to implement, and a choice has to be made between using stored pointers to chain the elements together (giving rapid insertions and deletions) or storing the elements consecutively, and moving them to handle insertions and deletions (giving slower and unpredictable processing time).

Stacks are used in the evaluation of expressions, in the dynamic linkage of subroutines, and in the FDP-11 for handling interrupts.

Queues are used for message and input/output request buffering, and for scheduling tasks at a uniform priority level.

2.6 Subroutine Linkage

The static connections between modules are made by the linker: this connects the 'call' statement in one subroutine with the 'entry' statement in another. This connection remains unchanged
during the execution of the program.

For the dynamic connection, in which the link is saved indicating which call is current, a standard convention is necessary. Most processors have a rudimentary convention built in their hardware: PDP-8 has its JMS instruction, PDP-11 its JSR, DDP516 its JST, all of which save in some standard place the contents of the 'next instruction address' register, and then jump to the specified destination. This is sufficient to save the current context within an assembly level program, but not for higher levels in which, for example, the index registers are not accessible to the programmer - so they must be saved and restored at subroutine entry and exit. Other conventions needed are how to pass parameter values into and out of the subroutine.

The nature of the subroutine call determines whether or not subroutines can be recursive. If each subroutine has one explicit place for its link, then it cannot be recursive. Thus the PDP-8 and DDP 516, which store the link at the start of the subroutine, cannot (using their hardware subroutine call instructions) deal with recursive subroutines. On the other hand, the PDP-11, which maintains a hardware controlled stack and pushes links onto this, can handle recursive subroutine links without trouble.

2.7 PDP-11 Subroutine Call

The PDP-11 subroutine call convention has developed since the machine was originally announced and the model 45 uses the stack even more extensively to permit subroutines to be reentrable.

SLIDE 2.10

The original form of the call with parameters used the words after the JSR instruction to contain the addresses or values to be transmitted. This is all right so long as either the addresses are all constant or the routine is not used reentrably, but not both. For if you want to reenter and an address modification is necessary, you cannot ensure that the version reentered has not had its addresses altered by another task. Reentrant subroutines must be read only. Therefore, the parameters to be passed to the sub-
routine must not be planted within the code. To unify with the treatment of recursion, they also are put on the stack, and suitable instructions performed to remain compatible with existing subroutine entry conventions.

SLIDE 2.11

All this entails appreciable additional instructions to be executed, and a messy unstacking operation at the return from the subroutine: hence a new instruction was created for the 11/45 (the MARK instruction).

SLIDE 2.12

An alternative method could have been used, of putting the parameters in the local work space of the subroutine rather than on the stack. This could reduce the amount of work to be done, but it would still be more than in the non-reentrant case.

Reentrant programming also requires a base for the task data to be held among the live registers of the processor. As yet, no conventions are established for this on the PDP-11, either in hardware or software. We can expect that registers 2 and 3 will become established as the bases for local and common task data areas.

2.8 Store Organisation

Another convention which the basic software establishes is the pattern of use of the main store of the computer. The linker forms a load module in a regular way, taking account of particular hardware constraints which might exist. Thus it will be aware of hardware pages corresponding to restricted ranges of addressing; byte/word/double word boundaries; and areas of addressing which have special uses (interrupt entries, peripheral controllers). It will be aware of the space occupied by a run time supervisor, and make the right links to that.

It will embody design decisions such as to have segmentation for separate concurrent programs with fixed size segments; to have overlays within programs in fixed positions; and not to have automatic page swapping because of the overhead and unpredictable response time.
A typical conventional storage layout is illustrated for the PDP-11 with DOS.

SLIDE 2.13

This shows the user area being a fixed region, in which program modules and common data space are assigned, and the stack given the remainder. A different layout is necessary to deal with a multi-tasking situation, when each task needs its own data and stack areas, and the programs must be guaranteed read-only.

2.9 Conclusion

This lecture has been concerned with the basic software, primarily that used for program preparation. It may appear that rather more is called for than is usually available with a small computer. This is sadly true in many cases, but the lack of provision by the manufacturers does not reduce the need of the users. Obviously a graded set of software is needed depending on the power of the computer (see Pyle and Langsford, 1970) and one would not expect powerful program preparation facilities to be available with a micro computer; however, one would expect a good set with a midi.

3. PERIPHERALS

3.1 Introduction

This lecture will seem rather like a catalogue - a list of all kinds of device which one might attach to a computer. The purpose of it is to show the range of devices, to classify them, and to point out some of the programming implications they carry. It also shows some of the peculiarities different devices have, justifying my earlier remarks on the need for device handlers. The lecture after this will treat device handlers in more detail.

The peripheral devices on a computer make the essential 'real-time' problem: each peripheral works at its own speed, has its own crisis time and frame repetition time. The nature of the disaster caused by exceeding the crisis time is not usually serious on the short time scale - for input devices, there may be an extended dead time when further data cannot be collected, and for output
devices there will be a reduction in the rate of data production. I give typical parameters with the devices, as a guide to the relative orders of magnitude. Serious disasters will more often be found at higher levels of programming, where there is a feedback loop involving an input and an output device, which must be serviced in some time dependent on the equipment beyond the immediate computer peripherals.

3.2 Classification

Peripheral devices can be classified according to the nature of their information source or sink, and for backing stores on the nature of the access sequences:

SLIDE 3.01

Interactive and real-time devices are either input or output - compound devices which carry data in both directions have distinct components for the two directions. Backing store devices usually can do output (main store to backing store) and input (backing store to main store) using the same device without distinct parts, but differ as to whether the backing store medium is addressable and hence randomly accessible. Many of these interactive devices are described and illustrated in the proceedings of the previous CERN Computing and Data Processing School (Zacharov 1971).

3.3 Interactive Input Devices

SLIDE 3.02

These range from a simple function button (usually replicated to correspond say to bits of a computer word), through the inevitable keyboard for character string input, to the various graphic input devices.

The light pen needs only one bit because it simply registers hit or miss on the currently displayed spot. The responsibility for identifying the coordinates of the required position is entirely with the software.

A rolling ball or joystick can indicate the direction to move a cursor on the display, and really needs a form of coarse/fine
control for convenient use. This can be obtained by two levels of angle discrimination on the joystick, or counting in a few bits for the rolling ball. The time gives a typical sampling time to follow a curved track. It is still necessary for the software to keep track of the absolute position of the cursor, taking increments from the device.

SLIDE 3.03

The tablet gives an absolute position of the stylus, so provides the most data directly to the computer. The x and y coordinates have to be read together, and the sample taken at about 20 ms intervals to be able to follow a curved track.

3.4 Interactive Output Devices

SLIDE 3.04

Although a lamp has only two states (on/off), it is usual to assign two bits to control an individual lamp, denoting switch on/switch off. For a group of lamps, one bit each may be used, up to a whole computer word. The whole group is put in the specified state whenever the computer word is written.

An incremental plotter needs only 3 bits, although some work using complete bytes for uniformity of communication.

Graphic displays need to specify x and y coordinates of individual points or increments for vectors. Storage displays have slightly less discrimination, and a longer settling time.

Characters drawn on a display are constructed from points or strokes. Basic displays use a 5 x 7 unit grid for each character, and construct the pattern to be displayed by hardware or software. As can be seen

SLIDE 3.05

dot matrix characters are harder to recognise but need less information to define them than stroke characters on the same grid. Hardware stroke generated characters are quite good.
3.5 Real-time Input Devices

SLIDE 3.06

Sensor relays and multiplexed relays are like function buttons, except that the time scale can be much more varied depending on the apparatus driving the relay.

Analogue-digital converters (ADC) come at different speeds and precisions with the orders of magnitude shown. A typical multiplexed ADC will have a control and status register to select the channel to be sampled, set up the gain appropriate for that channel, and then stimulate the basic converter to put the digital value into a data buffer register. This must be read by the computer before the next sample has to be taken.

Camac obviously can be included in this category, but discussion of that is deferred to subsequent lectures by Dr. Zacharov on systems aspects of small computers.

3.6 Real-Time Output Devices

SLIDE 3.07

Effector relays and multiplexed relays are similar to lamps in interactive devices. These are the essential components for Direct Digital control of equipment. The reason for the two bits for a single effector relay is to give positive indication when a change is needed. In the case of multiplexed relays, the word is written with all bits together, and the act of writing indicates that the states of all the relays have to be set approximately.

Incremental control of for example a stepping motor needs one bit to denote direction, and one bit to stimulate each increment or better two bits to start and stop movement.

All of these output devices would be associated with input devices to ensure that the controlled equipment had moved as commanded. The consequent feedback would have time-constants important for the stability of the control.

3.7 Backing Store Devices - Serial Access

SLIDE 3.08
It is not conventional to classify tape as a backing store, but it is often used for that purpose as much as for data transfer between a key-punch and a computer. However, it does depart from my criterion that input and output take place on the same device.

All these devices are accessed serially, even though blocks of data or DECtape can be individually addressed. The time to reach a given block depends directly on the difference between current and desired addresses.

Magnetic tape of the conventional industry - compatible sort can only be subjected to elementary position control: next or previous record, next or previous file, or start of tape.

3.8 Backing Store Devices - Random Access

SLIDE 3.09

Rotating magnetic devices come in many shapes and sizes. The different subunits of a disk are shown, not all of which are applicable to given circumstances.

The drum and fixed head disk are logically the same, differing only in their physical arrangement. Moving head disks have several cylinders, one for each possible head position. Demountable disk packs allow the volume of backing storage to be extended indefinitely by manual action, and also they allow the transference of data between systems (given compatible recording standards). Typical disk parameters are given below:

SLIDE 3.10

Most computer manufacturers do not make their own disk units, but buy in from peripheral manufacturers. Therefore you find the identical disk offered by different manufacturers, with just a different coloured box. The numbers given above are those for the FDP-11 disks. The RC11 and RF11 both have fixed head disks, with the RC11 using fixed sectors on each track but the RF11 writing words continuously round each track. The RK11C uses the top and bottom surfaces of a single platter, and the RP11 uses a stack of ten platters on a common spindle. The numbers of sectors per track
on these disks is very annoying, but the recording density could not manage 16 sectors in the innermost track.

3.9 Conclusion

In describing all these peripherals, I have mentioned the time scale as much as possible, rather than give specific hardware priorities as would be done in making a particular configuration. The best approach to deciding priorities is to determine the patience, and assess the amount of time needed to compute the response from each. From these, get the maximum permissible latency and assign priorities to give precedence to the devices for which this is the smallest.

3.10 Postscript

After presentation of the lecture, points were raised in discussion about a number of other peripherals. Some of those were specific to large computers, and are not appropriate to record here. Among peripherals relevant to small computers are:

Cassette tape (Standard Phillips cassettes, available on IBM System 7 and Phillips computers, also smaller computers from Burroughs, NCR). The facilities are virtually those of paper tape, with negligible positioning control.

Video tape (as used for video recording, with the sound track carrying block marks). This has slow start-stop and reverse direction characteristics, so must principally be used for continuous recording, such as high volume data acquisition.

Optical film exposed to display. Computer control of the frame advance and filter insertion can be used to make movie films and colour films. Also by varying the period of time for which the display is left on while a frame is being exposed, graded levels of intensity can be obtained on the picture.

- Large core store (now available quite cheaply)
- Read only memory (diode matrix used for bootstrap loaders)
- Associative memory (looking for a use)
- Holographic memory (rumoured but no firm information is available)
4. RUN-TIME SOFTWARE

4.1 Introduction

In my second lecture I described the run-time behaviour of a PDP-11 doing a subroutine call, but originally from the point of view of the instructions which had to be compiled for subroutine entry and exit. Now I want to go into more specific aspects of the run-time software, particularly addressing modes, task organisation and reentrant programming, interrupt handling and device handlers.

4.2 Addressing Modes

It is characteristic of small computers that their short word lengths can contain only short address fields, and consequently that addressing of the store may be awkward, with hardware restrictions on the regions which can be accessed from a particular location.

SLIDE 4.01

The PDP-8 instruction word has a 7 bit address part which permits access within a page of 128 words. A mode bit in the instruction word selects the page - either page zero or the page containing the current instruction. Thus only those two pages can be accessed directly. A further bit in the instruction word indicates indirect addressing, so the word addressed directly can be used (taking an extra storage cycle) to access to any word within the current store field of 4K. To get outside the 4K requires a separate instruction to change fields. Thus we see the restrictions of the address part and of the word length.

SLIDE 4.02

The example shows how words in page 0 (e.g. EXPNT, FRACT) have to be used for communication, and instructions in another page can refer to those or to words local to themselves.

SLIDE 4.03

The DDP516 is quite similar, but the longer word length permits bigger pages. Also it has an index register, so the address part
of the instruction need not be absolute but can be the offset from the word addressed by the index register. The mode bit selects page 0 or the current page; the indirection bit specifies whether the word accessed is to be used directly or as a pointer. Now it stops being like the PDP-8. If there is indirection, the pointer word can itself specify further indirection, and so on with a hardware limit at 8 deep (allowing an escape from 'indirect to self').

There are two modes which the processor can be in for address interpretation: normal mode (access within 16K) or extended mode (access within 32K). The difference is taken up by a bit which in normal mode denotes post-indirection indexing (while the index bit in the instruction word denotes pre-indirection indexing). In extended mode there is only the index bit in the instruction word, and that then denotes post-indirection indexing.

SLIDE 4.04

The example shows the indirection bit (minus sign on the assembled word) and the indexing bit, (leading 1 in the assembled word) and absolute references to page zero (e.g. wd 14) as well as relative references to the current page (e.g. wd 13).

SLIDE 4.05

The Modular One has more addressing modes. The address part of the instruction word gives an 8 bit offset within a 256 word page. There are three segment bases, three registers which can be used as modifiers, and the possibility of indirection, but not all combinations of these. The three-bit mode field, E, distinguishes which combination of base and modification is to be used, from literal use of the address part as an 8-bit immediate operand, to the X segment (which contains the instructions, current instruction position is P) or the Y or Z segments for data. The interpretations given apply when the processor is in normal user state; there are some differences when it is in supervisor state. If there is indirection (modes 3 or 7), then the word accessed is itself interpreted with several modes: a different set of 8 modes apply for the pointer word.

SLIDE 4.06
The PDP-11 is very different. It uses its 16 bit word to contain two addresses in many instructions, but certain modes take an additional word after the instruction. Each address part consists of just 3 bits with a 3 bit mode selector. So the instruction in the first instance refers to one of the eight registers in the machine, but the modes allow these to give addresses of words in store and even use these indirectly. There are no page boundaries, and no further mode bits in pointer words. The modes include indexing (modes 6 and 7) in which the offset for the register occupies the following word, and automatic incrementing and decrementing of the register, which implement both the processor stack (on R6) and constant operands for instructions (by incrementing modes on R7 with the operand in the word following the instruction).

SLIDE 4.07

The example shows how instructions are constituted, with the mode 0 frequently used to designate the register as operand (e.g. destination address in word 116) and mode 2 with register 7 used to get the following word as operand (e.g. source address in word 116), also the stacking modes (e.g. in words 46, 56).

4.3 Task Organisation

As we have noted, small computers have a dedicated job to do, handling the data to and from a number of devices, and not (usually) general purpose computing. The devices which produce and absorb the required data will be unsynchronised with respect to one another, and although each may require a fairly simple sequence of computer operations, the interleaving of all of these can become very involved to deal with all the different activities. It is possible to design a program which will look after everything explicitly, by treating all the devices as artificially synchronous on a very short time scale, and making explicit tests to interleave the slices of the different operations.

SLIDE 4.08

It is conceptually cleaner, much easier to understand and test, and simpler to program (although possibly with a greater overhead),
if we define each logically separate sequential action as an independent task, containing 'wait' indications at appropriate points where there is nothing further at the time to be done for that task. The tasks are then run under the control of a supervisor program which schedules the activity of the processor between the tasks. The execution of each task (which is called a process) can then be considered as asynchronous with respect to the other tasks, although at a microscopic level (e.g. in the supervisor) the separate parts do have a strictly determined time sequence. This approach separates what is fundamentally essential — the sequence of activities for one device — from that which is incidentally necessary using a single processor.

If there are several devices of the same type working concurrently, then we have to consider reentrant device handlers.

4.4 Reentrability

We already have the idea of a subroutine or procedure which can be activated, possibly with parameters, by a single statement in a calling routine. When the subroutine finishes, the calling routine continues after the calling statement.

A recursive subroutine is one in which the calling routine can be the same as the subroutine. From within its body, it can call for a new activation of itself. The problem of run-time organisation for recursive subroutine is to make sure that the local variables of the various activations of the subroutine are kept distinct. (There is also a problem of passing parameters.) Because of the nesting character of the activations, it is satisfactory to keep the local variables on a stack. Each activation needs a new set of local variables on the stack. While it is in operation, other (calling) activations are not moving — they are still at the calling statement. When an activation finishes, its local variables are freed from the stack, and its calling activation can resume.

A reentrant subroutine is more complicated than this, and cannot use a stack to hold its local variables. A reentrant sub-
routine is one for which there can be different activations arising independently and asynchronously with its own progress, with the possibility that new activations must be begun before previous activations have finished. This is typically the case for device handlers. If the subroutine does not contain 'waits' there is little point in starting a new activation rather than letting the current activation finish; but if there are 'waits' then we must begin a new activation. As with recursive subroutines, the problem for the run-time organisation is to maintain distinct sets of local variables for the various activations. But because there is no necessary relation between the starts and finishes of activations, the order in which the sets of local variables are freed is not the reverse of the order of assignment, so they cannot be stacked. A list structure is needed for them, in which deletions can be made in the middle. Each activation then needs to have a distinct pointer to define the base of its own activation, and whenever there is a task switch which can change the activation, this pointer must be changed - so it must be one of the live registers.

The rules of reentrant programming are therefore to ensure that the program and data are absolute distinct, than the program is absolutely unchanged during execution, and that all the local data is always specified relative to a base in a live register. Creation of the space for the local data and putting the right value in the base register are the responsibilities of the task initiator and scheduler. Thus in the case of a device handler, when the device is initialised, it is necessary for the initiator to assign buffer space and local work space for this particular device, and notify the scheduler of both the entry point to the reentrant device handler, and the base for data space for this particular activation of it.

4.5 Interrupt Handling

We have discussed previously the general topic of when to make the processor calculate a response to a stimulus, now we discuss what it does, for some typical cases.
To input a character from a keyboard or paper tape reader, we do the following:

Similarly to output a character, we 'take out' instead of 'put in', and test for buffer empty. For a disk or tape transfer, we get the next command (seek, search, read block, write block etc.) from a list set by the user, decode it, and set the command and status register appropriately. Each of these will give rise to an interrupt from the input/output controller denoting completion, upon which there is a further status check.

It is not hard to see that unless we are careful the amount of processing to be done in the interrupt routine can become rather large, and this puts at risk the whole interrupt regime, because it extends the period when interrupts are inhibited, and lengthens the latency time of the processor.

We deal with the problem by careful discrimination about what is critical in the microscopic real-time sense. Usually the transmission of data in or out and resetting the status are all that are essential for the apparatus to continue. Checking of the data input and consequent actions or other interpretations are not quite so critical, in that they need not be done before the apparatus is re-engaged, although they must be done before the next unit of data arrives. We split the processing into first level and second level sections, with first level and second level interrupt handles and a low level scheduler.

The first level interrupt handler (FLIH) does the critical action, and makes an entry for the required second level interrupt handler (SLIH) in the scheduler list. At a lower priority than all the FLIH's, the scheduler scans its list and activates the specified SLIH's. Usually the scheduler list is empty (otherwise the system is overloaded).

4.6 Device Handlers

As I said in the lecture on peripherals, each peripheral device
has distinctive characteristics of status and control, so its input/output controller needs a distinctive response from the processor, and therefore a distinctive device handler to make it work. To illustrate the variations let us examine the control and status registers for some peripherals on the PDP-11. There is considerable similarity between them, but even with the systematic approach which the PDP-11 laudably has, the differences are also apparent.

SLIDE 4.11

For the simple devices (keyboard, KBS; teletype TKS; paper tape reader PRS; line printer LPS), the effective bits are subsets of a general pattern— but some have an 'enable' or 'go' bit, others do not; some have an 'error' indication (X), other do not.

For the disks, the situation is much worse. More registers are of course needed:

- Word count WC
- Bus address BA (usually address of data block in store)
- Disk address DA
- Current Disk Sector DS
- Error Information ER
- Maintenance MN

as well as the overall control and status register. But because of different track/cylinder sizes, the use of the DA register is different; likewise the ER and MN registers are different. The Control and Status registers are partly similar (especially the low byte and the error bit), but the various error indications and additional status in the high byte are unrelated. Thus the device handlers for these devices have to be separate routines, to give the same facilities to the user program.

We can take the standardisation a stage further in an interactive system, to get a standard method of taking the user's input on a keyboard will echoing on an associated output device. We
should echo his input back 0.1-0.2 sec, with a program controlled
inhibition of the echo for passwords etc.

SLIDE 4.12

The control actions include end of message, delete last character,
delete line, and for a display also the cursor movements. The end
of message action could be to pass the buffer contents to another
level of analysis - perhaps a command interpreter.

4.7 Conclusion

In this level we have been considering the detailed action of
the computer during execution of a program, from the hardware interpre-
ation of the addresses (and the differences between several
computers in this) to the way we organise the software immediately
next to the input/output controllers. This led us to an apprecia-
tion of the need to separate tasks, and make the processor of the
computer drive several processes concurrently. In the next lecture
we will think more about multi-process systems, and especially
those involving more than one processor.

5. MULTI-PROCESSOR SYSTEMS

5.1 Introduction.

The small computer in physics began as an isolated data
processing device attached to an experiment. Soon people found
they had to take the data it had collected, and subject these data
to further processing. So the problem of data transport arose, and
we had problems of magnetic tape compatibility, different word
lengths, and different number formats.

However, the physical carrying of a magnetic tape was clearly
undesirable, so we started thinking about data links, so that the
data could be sent directly from the small computer to the large
computer.

From this point on, we realised that the system of interest
was no longer just the experiment with its local computer, but the
whole establishment with its network of computers. Thus we came to
have a multi-processor system, comprising a number of interacting
computers.

As soon as we have a network and we realise the importance of our data communication, we worry about the reliability of our network control computer, and think that perhaps we should have a multi-processor system there, comprising of two processors which can cooperatively or separately handle the network.

Thus the physicist comes to be interested in multi-processor systems.

5.2 Definitions

SLIDE 5.01

We examined in the previous lecture the reasons for expressing a real time program as a set of tasks, which are performed concurrently as several processes. If we have only one processor to drive them, then obviously we need some software mechanism for scheduling its time between the various tasks - this is the supervisor. Not so obviously, if we have several (physical) processors which can drive them, we still need a software mechanism to schedule the time of this fixed number of physical processors between the potentially variable number of tasks. Consequently, if one of the processors stops working for one reason or another, the mechanism is already there to permit the remaining processors to continue driving the system. Note that for this continuity to be possible, the processors must be assignable indifferently to the tasks which have to be done. Particularly, this means that they must all be able to access the store containing the programs and data for the tasks.

5.3 Protection

If there are several tasks in the same store, then some mechanism is necessary to ensure that accesses belonging to one task are restricted to the regions of the store relevant to that task.

We can identify several objectives which protection aims at achieving - preventing various forms of unauthorised access.

SLIDE 5.02

The methods which are used to implement protection recognise
that store is used in contiguous areas over which uniform protection is required: i.e. segments. It is then a matter of programming to define the specific areas (in some quanta of perhaps pages rather than words) and define the protection required for each area. The hardware has to apply the tests to detect infringement, based on the criteria specified by the software. Because the definition of the uniformly protected area is closely tied up with this, a hardware relocation mechanism naturally belongs with protection.

This in fact gives an even stronger form of protection, through the introduction of a mapping from virtual address space to physical address space. The effective addresses of all instructions executed are within virtual address space, and if this space does not cover part of physical address space, then there is no means by which an instruction executed can even refer to a word in that part, let alone refer but he prevented by a test for unauthorised access.

When the need for protection was first recognised, the principal worry was about garbling. (This is still the principal worry, and the reason why people are unwilling to develop new programs on a computer while it is running on-line.) Consequently two modes of access were distinguished, ordinary and write-inhibit. In ordinary mode, words in the segment could be read as data or instructions for execution, or written (as data). In write-inhibit mode, words could be read as data or instructions, but an attempt to write was prevented. This is the distinction still made on most machines which have protection at all.

SLIDE 5.03

Two more elaborate protection schemes are those of the Plessey System 250 computer (England, 1972) and (quite different) of the FDP-11 model 45.

The System 250 distinguishes read accesses for instructions from read accesses for data, so that for example a segment can be execute only, capable of being performed but not examined. It also gives further special protection to the segments in which these access control and relocation data will be held. Of course, while a processor is active it has the access control bits in live
registers, but when its task is switched, these have to be stored and replaced by those for a new task. The segment in which this information is stored is much more sensitive than a segment containing ordinary data, so separate modes are introduced to guard it. The access control and segment location information is collectively called 'capability'. A capability segment is one containing such information, and access can be made to it only if the appropriate capability mode is set up.

The PDP-11/45 also distinguishes instructions from data, but by different addressing spaces rather than different access control. There are separate mappings from virtual to physical address space, depending on whether the processor is doing an instruction fetch or an operand fetch. Thus we have I-space and D-space. These are not necessarily distinct, and indeed to make the subroutine call mechanism work, the stack has to be identically in both I and D space (since it contains instructions to be executed as well as data).

Each address space, I and D, consists of eight separate segments, each up to 4K words long in quanta of 32 word pages. Each segment has its own access control mode, which determines what sorts of accesses are permitted to it, and what is to happen on infringement.

SLIDE 5.04

Not only does the 11/45 permit or prevent accesses, but it has an intermediate monitoring mode in which the access is allowed to take place, but is followed immediately by a trap so that recording or investigating action can be taken. The combinations of possibilities for read and write access to each segment are shown in the following table, which gives the value in a three bit control field. (I space is treated the same as D space, although there should never be any write access to it.)

The PDP-11 model 40 has a subset of these modes which coincides with the traditional form of protection: there is only one mapping, not distinguishing I from D space, and none of the trap modes are
implemented. This leaves three access modes: permit, read-only and no access.

Before leaving protection, I must mention two problems which it raises, which have not yet been satisfactorily solved. First the cost of the overhead—we have more live registers, so an additional burden for task switching. The second problem is more tricky: what protection is to be applied when an input/output controller is doing a block transfer. One approach is to have the supervisor check before initiating the device handler, and then let the device handler work with physical addresses. Another approach is to treat the input/output controller as another processor, and make it do its own access checking. This issue is far from settled.

5.4 Multiprocessor Configurations

In my first lecture about computer architecture, I explained about bus and net configuration topologies. The same issue comes in more conspicuously when there are several processors. Contention for store access is no longer an occasional problem arising during block transfers, but an every microsecond fact of life, as accesses to the store are made by both processors for instructions and data. Each store module has to have its own access controller to resolve the contention, on the basis of fixed priorities between the processor.

Separate processors may have some store physically attached to themselves, and other store in common.

SLIDE 5.05

This would permit, for example, common programs to be used reentrantly by both, with separate data areas in the private stores. If either processor stopped, the tasks whose data are in its private store would then be lost. Alternatively, the data for the tasks could be held in the common store, and separate copies of the programs held in the private stores. This does not economise on storage needed, but permits either processor to take on all tasks if one processor fails.
If we really must be able to resume all tasks after a breakdown, then a configuration like this is needed:

**SLIDE 5.06**

Each processor has a separate bus, which it controls. Each store module is connected to every bus, and has its own access controller.

The actual processor driving any particular task is irrelevant, and may change during the course of that task. A similar method is used in the Hughes Aircraft computer with floating executive control (Parisier 1964).

All this is very interesting, but not likely to come to physics for some time yet, apart from message switching and computer network control.

**5.5 Multi-Computer System**

More realistically, we do not have shared store. Rather, we have two processors each with its own private store, and its input/output controllers, so that each comprises a separate real-time computer.

**SLIDE 5.07**

The computers directly interact with each other, via their input/output controllers, so that each appears to the other as a "funny" peripheral. It is particularly funny, because not only can it respond to stimuli, but it can spontaneously initiate an interaction.

Because of this possibility, there is a difficulty of defining the idling state. After having fully handled one interaction, both computers have to be set in a state of being receptive to initiating stimuli from the other, but capable of having their own state changed internally when they find they need to initiate an interaction. (The initiative can be for a data transfer in either direction: 'send data' or 'request data'. The difference between these is not great, however, because there is always information passed in both directions to define status.)
5.6 FDP-8 linked to System/360

As an example of this, I will describe the processing problems for a link between a FDP-8 and a System/360 which we developed at Harwell. The hardware was a locally designed FDP-8 adaptor and the IBM supplied 2701 with Parallel Data Adaptor. The computers are less than 50 feet apart, so there are no serious data transmission problems.

SLIDE 5.08

The 2701 interfaces allows an ATTN signal to come in, and READ and WRITE signals to go out. The FDP-8 adaptor was made to generate an ATTN signal under program control, and to cause distinct interrupts on receiving a READ or a WRITE signal. Data is transmitted in 16 bit words. The FDP-8 can select one of two ways of using these - either 12 bit FDP-8 words map onto the lower 12 of these 16 bits, or a pair of consecutive FDP-8 words can map onto the two constituent bytes.

If the FDP-8 has data to send (and the data may be intended to be interpreted in the 360 as a request for some reply, but that doesn't matter as far as the link is concerned), then it first sends the ATTN signal. The 2701 receives this, and interrupts the 360. The 360 program detects the source of the interrupt and enters the Harwell-written attention routine for this 2701. That routine sets up a standard channel program to read a record of 80 words (i.e. 160 bytes) from this 2701, and executes a Start Input-Output instruction. The Input-Output Controller (Channel) on the 360 then sends the read command to the 2701 which causes it to assert its READ signal. This is received by the FDP-8 adaptor, and now makes a read-interrupt in the FDP-8. The FDP-8 is of course expecting and waiting for this, so it is ready to perform a block transfer of the data it wants to send. The words of data are sent autonomously, and after the last word (normally there are less than 80), the FDP-8 asserts end-of-record, which the 2701 recognises and makes the channel program finish, causing a normal I/O complete interrupt on the 360. Now the 360 has the data and begins processing it.
For a transfer in the other direction, the 360 spreads the desired message into alternate bytes of a 160 byte buffer, sets up a channel program to write the buffer to the 2701, and executes a start Input/Output instruction with no prior warning to the PDP-8. The 360 channel commands the 2701 to assert its WRITE signal. The PDP-8 adaptor receives this, and makes a write-interrupt in the PDP-8. The PDP-8 program knows that this is likely to happen without warning, and that it means an 80 word record is coming. So it makes a buffer space for 80 words, and sets itself up to perform a block transfer into this buffer. When the block transfer is complete, there is another interrupt in the PDP-8, so that it can deal with the data it has received.

The 80 word records are always of a standard format, using only the low 8 bits of each word, with an 8 word header and up to 72 words of text.

Difficulties arise if both computers decide that they want to do a transmission at the same time. Then we have to decide a convention, and build into the programs, that one of the computers will give way, and yield to the precedence of the other. We make the 360 give way, so that the PDP-8 has precedence on the link.

The argument for this is like 'steam gives way to sail'. The 360, being more powerful, is more capable of handling the additional logic of giving way and resuming afterwards. The 2701 hardware also helps, since in the situation we envisage it has asserted WRITE but received ATTN. Its action in this case is to cause an abnormal channel end with a truncated channel program. The 360 responds to the situation by servicing the attention in its usual way with a read channel program, leaving its own message until later. In the PDP-8, after it has sent the ATTN signal, it ignores any write-interrupt which comes, and just waits for a read-interrupt.

5.7 More than two computers

We have discussed up to this point a system with only two computers, that is, one link. If we have several computers there are configuration problems again, and the simplest form of arrange-
ment is a radial configuration, in which one computer is special, at the hub, and the others are all connected to it.

SLIDE 5.09

Now the hub computer is in control of the switching of links. It may not have the capacity to handle all the rim computers transmitting concurrently, so it has to switch its data path to a selected rim computer. However, when a rim computer wants to send, it must be able to signal the hub computer even though the data path is not connected to it. Therefore we see we need control paths from all the rim computers to the hub computer, and a switchable data path between any rim computer and the hub computer.

Such a system has been designed and implemented at Harwell (Fergus 1972). A standard protocol for the conversation between the source and destination is then needed.

SLIDE 5.10

The processing in the transmitting and receiving computers includes checking the format of the prologue, and listening out for an error at the end.

SLIDE 5.11

5.8 Conclusion

We have come a long way from high energy physics now, but the techniques of making computers communicate with other computers are not vastly different from those of data taking and control. This subject now leads into the matter of data links, which will be covered by Dr. Zacharov's lectures on Systems Aspects of Small Computers in Elementary Particle Physics.
REFERENCES


Calderbank, V. J. and Calderbank, M. 1971 Private Communication.


Without operator intervention, can accept data, emit results, process data under control of an internally stored program, respond to externally generated stimuli.

Slide 1.01 Computer (digital, real-time)

Cost — £2–20k

Use — Dedicated

Size — < 1 cabinet

Structure — Simple, Modular, Extendable

Word length — Short

Instructions — Few, simple

Slide 1.02 Small Computer Characteristics
<table>
<thead>
<tr>
<th></th>
<th>Micro</th>
<th>Mini</th>
<th>Midi</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cost £k</strong></td>
<td>3–5</td>
<td>4–6</td>
<td>7–15</td>
</tr>
<tr>
<td><strong>Word length bits</strong></td>
<td>8–12</td>
<td>16</td>
<td>≥16</td>
</tr>
<tr>
<td><strong>Store size Kw</strong></td>
<td>≤4</td>
<td>4–32</td>
<td>8–64</td>
</tr>
<tr>
<td><strong>Application</strong></td>
<td>Monitor/Control</td>
<td>Monitor + Control + some dp</td>
<td>Several Mon + Con with dp</td>
</tr>
<tr>
<td><strong>Variability</strong></td>
<td>Rare</td>
<td>Manual</td>
<td>Automatic</td>
</tr>
<tr>
<td><strong>Peripherals</strong></td>
<td>Few</td>
<td>Several</td>
<td>Many</td>
</tr>
<tr>
<td></td>
<td>Scientific</td>
<td>Commercial</td>
<td>Immediate</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>Processing</td>
<td>Numerical</td>
<td>Organisational</td>
<td>Structural</td>
</tr>
<tr>
<td>Input volume</td>
<td>Small</td>
<td>Large (files)</td>
<td>Large (data)</td>
</tr>
<tr>
<td>I/O types</td>
<td>Few (pt, cr, lp)</td>
<td>More (mt, disk)</td>
<td>Many (special)</td>
</tr>
<tr>
<td>User I/O</td>
<td>via off-line device</td>
<td>On-line</td>
<td></td>
</tr>
<tr>
<td>Response time</td>
<td>Minutes — hours</td>
<td>Milliseconds — seconds</td>
<td></td>
</tr>
</tbody>
</table>

Slide 1.04  Computer Applications

Slide 1.05  Small Computer Structure
Event occurs

Processing switched to appropriate task

Give reply

Disaster happens unless a reply has been given

Next event occurs

Slide 1.06  Crisis Time
Slide 1.07 Interrupt Process

I/O Controllers
- Single Word
- Block Transfer

A D Data Channel

Store

C: Control
S: Status
D: Data
A: Address

CS

D Slow Device

Fast Device

Slide 1.08 Input/Output organization (including Direct Memory Access)
A  Address
C  Control + timing
D  Data
R  Request bus mastery
G  Grant bus mastery
Slide 2.01  Program Preparation

Visibility of instructions
Explicit  Embedded  Implicit

Awareness of hardware constraints
Full  Moderate  Slight

Software conventions
None  Few  Several

Slide 2.02  Programming Languages
: PDP-11 DISK MONITOR INITIALIZATION ROUTINE
: THIS ROUTINE IS LOADED BY THE BOOTSTRAP ROUTINE
: AND CONTROL IS GIVEN TO IT AT THAT POINT
: THE FOLLOWING TABLES ARE INITIALIZED:-
: DDL
: MRT
: VECTORS
: STACK PROTECTION
: AT SUCCESSFUL TERMINATION, THE MONITOR
: IS READY TO ACCEPT A KEYBOARD COMMAND.
: INITIALIZATION BEGINS HERE.

BG.BGN: MOV PC,SP ;SET STACK POINTER
TST -(SP)
MOV #200,@#177776 ; INITIALIZE STATUS
MOV #EMT,.@#30 ; SET EMT VECTOR
MOV #340,@#32
MOV #BG.BET,.@#4 ; DETERMINE CORE SIZE
MOV #40,@#6

MOV #40000,R1 ;OK
BG.ETS: TST (R1) ; POSSIBLE FAIL
ADD #20000,R1 ; TRY 4K MORE
BR BG.ETS
BG.BET: MOV R1,R0 ;USE CORE SIZE TO ...
SWAB R0 ; ... DETERMINE BUFF SPACE
ASR R0
ADD #BF,STH,R0 ; (HALF AVAILABLE)
MOV R0,BFE ; SET TABLE END ACCORDINGLY
CLR (R0)+ ; LEAVE BAT LINK WORDS
CLR (R0)+
MOV #EOM,R2 ; ALSO ADJUST MONTOP

MOV R0,(R2)+ ; ... & STACK TOP
ADD #40,R0
MOV R0,(R2)+
MOV #107070,R0
TST -(R1)
MOV R1,R2 ; ... AS LAST WORD ADD:
MOV #DAT,R0 ; NOW PICK UP DATE & TIME
CLR8 R1
MOV (R1)+,(R0)+
MOV (R1)+,(R0)+
MOV (R1)+,(R0)+

Slide 2.03 PAL-11
PROCEDURE SCALEFACTOR(R5):
BEGIN COMMENT ADDS TO THE SCALEFACTOR IN R2:
  NEXTCHAR: IF R0 = "-" THEN
  BEGIN SET(EXPOSIGN); NEXTCHAR;
  END FIRST RESET(EXPOSIGN);
R1 := R0 AND #F; NEXTCHAR;
WHILE R0 >= "0" DO
BEGIN R0 := R0 AND #F; R1 := R1 * TEN + R0; NEXTCHAR;
END;
TEST(EXPOSIGN);
IF = THEN R2 := R2-R1 ELSE R2 := R2+R1;
END ;
COMMENT BEGIN INSsymbol;
LA(R0)(0); IC(R0)(CBUF(R6));
TOP:
MVC(1)(VALUE)(BLANK); LA(R2)(0);
WHILE R0 = " " DO NEXTCHAR;
IF R0 = "-" THEN
BEGIN SET(SIGN); NEXTCHAR; IF R0 < "0" THEN
BEGIN LA(XR)(0); ERROR; GOTO TOP;
END;
END ELSE RESET(SIGN);
IF R0 >= "0" THEN
BEGIN COMMENT READ A NUMBER;
  R1 := R0 AND #F; R2 := 0; NEXTCHAR; MV1(1)(TYPEFLAG);
  WHILE R0 >= "0" DO
  BEGIN IF R2 > 7 THEN IF R1 >=214748364 THEN
  BEGIN IF "-" THEN
  BEGIN LA(XR)(19); ERROR; R1 := R1 - R1;
  END ELSE IF R0 > "?" THEN
BEGIN LA(XR)(19); ERROR; R1 := R1 - R1;
END;
R0 := R0 AND #F; R1 :=R1 * TEN +R0; R2 :=R2+1;
NEXTCHAR;
END ;
COMMENT INTEGER IN R1, NEXT CHARACTER IN R0;
calculate wages

BEGIN COMMENT This programme fills the table 'record' with whole word integers & then, using information drawn from elements of the table & further information on hours worked from tape, calculates the wages of each employee in £ s d.:
INTEGER i<std,th,pence,shill,payl,pays,payd;
(i - count variable; std - hours at standard rate; 
time + 1/2; Pence, shill - workspace; 
payl,pays,payd - result
TABLE record[1,200]

[number UNSIGNED (9) 0,12;
deat UNSIGNED (3) 0,9;
rates UNSIGNED (5), 0,4;
rated UNSIGNED (4)0,0];
(for input information)

PROCEDURE div(VALUE INTEGER a,b; LOCATION INTEGER c,d);
BEGIN COMMENT this procedure performs entire division:
 c := 0;
 FOR c := c-1 WHILE a > 0 DO a := a-b
 c := c-1; (entier)
 d := a+b; (remainder)
END of div;

FOR i := 0 STEP 1 UNTIL 199 DO
BEGIN COMMENT this loop repeated for each employee:
 read(record[i]); (read info. for i’th employee)
 read(std); (read hours at standard rate)
 read(th); (read hour at time + 1/2)
 Pence := rated[i]*(std+th*3/2);
 div(pence,12,shill,payd);
 (convert Pence to shillings and Pence, putting remainder as Pence figure of result)
 shill := shill+rates[i]*(std+th*3/2);
 div SHILL,20,payl,pays); (convert shillings to pound and shillings as result)
 COMMENT Now print results:
 print(number[i]); print(dept[i]);
 print(payl); print(pays); print(payd)

END of loop

END of programme

FINISH
; P = SET PRINT MODE
; P: IF CHARIN-ZERO.LT.0.GOTO Z; READ MODE DIGIT. CHECK
    IF ( ).GT.0:GOTO Z;
    ( ) =⟩PMODE; NEW MODE
    GOTO COMMAND;
;
; R = SET RELATIVE ADDRESS
; R: NUMIN(AMODE) =⟩RELADDR; READ ADDRESS
    GOTO COMMAND;
;
END;
---------------------------------------------------------------

PROCEDURE RAR;
; READ ADDRESS RANGE, LEAVING RESULT IN LOWADDR, HIGHADDR
    NUMIN(AMODE)+RELADDR =⟩LOWADDR; READ ADDRESSES
    NUMIN(AMODE)+RELADDR =⟩HIGHADDR;
    IF ( ).LT.LOWADDR:GOINDTO AERROR; CHECK VALIDITY
    RETURN;
END OF RAR;

;---------------------------------------------------------------

PROCEDURE MESSOUT(STRING)
; OUTPUT MESSAGE STRING. MESSAGE PACKED 1 CHAR PER WORD IN STRING.
; PRECEDED BY NO. CHARs IN STRING

INTEGER INDX;
    LOOP(1,UNTIL.STRING(O)) =⟩INDXX;
    CALL CHAROUT(STRING(INDXX)); OUTPUT EACH CHARACTER
    ENDLOOP;
    RETURN;
END OF MESSOUT;

---------------------------------------------------------------
Slide 2.08  Software Conventions  (Array and Character String)
Array  Fixed size, Random access

SQL  Variable size (fixed maximum)
     Ordered access

Stack.  Put in and take off same end

Queue.  Put in and take off opposite ends

List.  Put in and take off anywhere
JSR R5, SUB
BR CONT
.WORD A
.WORD B
.WORD C

SUB: MOV B R5,N
MOV @2(R5),R1
MOV @4(R5), R2
body
RTS R5

CONT:

Stack contents on entry to subroutine

Stack

R6 → old R5

Slide 2.10  PDP-11 Subroutine and Calling Routine (1)
MOV R5, -(SP)
MOV JSBR, -(SP)  JSBR: JSR R6 @ R5
MOV # C, -(SP)
MOV # B, -(SP)
MOV # A, -(SP)
MOV BRN, -(SP)  BRN: BR. +8
MOV SP, R5
JSR PC, SUB
RET: MOV [(SP) + , R5

Stack

old PC

RET
BR. +8
A
B
C
JSR R6, @ R5
old R5

Stack contents on entry to subroutine

Slide 2.11  PDP-11 Calling Routine (2)
MOV R5, -(SP)
MOV # C, -(SP)
MOV # B, -(SP)
MOV # A, -(SP)
MOV MN, -(SP)    MN: MARK 3
MOV SP, R5
JSR PC, SUB
RET:

Stack

<table>
<thead>
<tr>
<th></th>
<th>RET</th>
<th>(old PC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R6</td>
<td>RET</td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>MARK 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>old R5</td>
<td></td>
</tr>
</tbody>
</table>

Stack contents on entry to subroutine
Slide 2.13  PDP-11 Storage Layout
Interactive (with human) \{ \text{In or Out} \}

Real-time (with apparatus) \{ \}

Backing store \quad \text{Out and In}

Serial access

Random access

\begin{tabular}{|l|l|l|}
\hline
Function button & 1 bit & 100 ms \\
\hline
Keyboard & 8 bits & 100 ms \\
\hline
Tablet & 20 bits & 20 ms \\
\hline
Light pen & 1 bit & 20 ms \\
\hline
Ball \quad & 6 bits & 40 ms \\
\hline
Joystick \quad & \}
\hline
\end{tabular}

Slide 3.02 Interactive In
Slide 3.03 Tablet

<table>
<thead>
<tr>
<th></th>
<th>Lamp</th>
<th>Type</th>
<th>Line print</th>
<th>Plotter</th>
<th>Alpha display</th>
<th>Graphic display</th>
<th>Storage display</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 bits</td>
<td>8 bits</td>
<td>8 bits</td>
<td>3 bits</td>
<td>8 bits</td>
<td>20 bits</td>
<td>18 bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>100 ms</td>
<td>10 ms</td>
<td>20 ms</td>
<td>20 μs</td>
<td>80 μs</td>
</tr>
</tbody>
</table>

Slide 3.04 Interactive Out
5x7 Dot matrix
35 bits/character

Strokes on 5x7 grid
Up to 20 strokes/character
4 bits/stroke

Sensor relays 1 bit 20 μs – 200 ms
Multiplexed relays 16 bits 100 μs – 100 ms
Analogue – Digital Converter 10–18 bits 30 μs – 5 ms

(Camac later)

Slide 3.06 Real-time In
Effector relays 2 bits
Multiplexed relays 16 bits
Incremental control 3 bits
Digital –
  Analogue
  Converter 8–12 bits

Serial Access  Position Control
Paper tape None
Magnetic tape Elementary
DEC tape Direct

Useful for data transfer between systems, as well as storage in one system.
Slide 3.09  Backing Store - Random Access
Rotation time | 25 ms | latency
--- | --- | ---
Surface selection | 300 µs | switch
Change cylinder | 15–150 ms | seek

<table>
<thead>
<tr>
<th></th>
<th>RCII RS64</th>
<th>RFII RS11</th>
<th>RK1K RKO2</th>
<th>RP11 RPO2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surfaces</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>Tracks/surface</td>
<td>32</td>
<td>128</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>Sectors/track</td>
<td>64</td>
<td>2048</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>Words/sector</td>
<td>32</td>
<td>128</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>Word trans time</td>
<td>16 µs</td>
<td>16 µs</td>
<td>20 µs</td>
<td>10 µs</td>
</tr>
</tbody>
</table>

Slide 3.10  Typical Disk Parameters
Slide 4.01  PDP-8 Addressing

/FLOATING POINT I/O ROUTINES
/REQUIRES FLOATING POINT INTERPRETER
/ENTRY AT 0007

0007  5600  FPNT,  5600
*7

0044  0000  EXPONT,  0
0045  0000  MORDER,  0
0046  0000  LORDER,  0

0052  0000  FPAC1,  0
0053  0000  0
0054  0000  0
0055  7777  SWIT1,  7777
0056  7777  SWIT2,  7777
0057  0000  CHAR,  0
0058  0000  DSWIT,  0

*52

0676  0000  PRCHAR,  0
0677  1056  TAD SWIT2
0677  7650  SNA CLA
0677  5767  JMP I PRCHAR
0677  1377  TAD LFED
0677  4776  JMS I OPUT
0677  5767  JMP I PRCHAR
0677  7344  OPUT,  OUT
0677  0212  LFED,  0212

Slide 4.02  PDP 8 Instructions
Honeywell

HONEYWELL INFORMATION SYSTEMS LTD

* ASR/KSR DRIVER

OLERT DOCUMENT NO. 06-028-000-690623

REL

* KB - START ASCII DEVICE ON I/O BUS

* X CONTAINS DEVICE I.D. BASE

00000 0 15 00777 KB  STX  SDT+5  SAVE DEVICE I.D. BASE
00001 1 02 00040  LDA  GDIB+1  ADR OF INTR BIT
00002 0 04 00772  STA  SDT  SAVE
00003 -0 02 00772  LDA*  SDT  FETCH INTR BIT
00004 140401  CMA  ONES COMPLEMENT
00005 -0 03 03335  ANA*  IHL1  MASK OUT INTR BIT
00006 -0 05 0772  ERA*  SDT  ADD INTR BIT
00007 -0 04 03335  STA*  IHL1  SAVE CONN INTR MASK
00008 -0 03 0337  ANA*  IHL4  AND WITH CRT INTR MASK
00009 74 0020  SMK  '0020  SET NEW HARDWARE MASK
0000A 1 02 00042  LDA  GDSS+1  FETCH OCP TO START DEVICE
0000B 0 04 00015  STA  ++2  SAVE FOR EXECUTION
0000C 1 02 00003  LDA  GDSC+1  FETCH SAVE C
0000D 0 00 00000  ***  **  OCP TO START DEVICE
0000E 0 04 00000  STA  0  SAVE IN X
0000F -0 10 00341  JST*  CA9  FETCH MODE ADR
00010 0 04 00772  STA  SDT  SAVE MODE ADR
00011 0 02 00772  LDA*  SDT  FETCH MODE
00012 0 35 00777  LDX  SDT+5  RESTORE DEVICE I.D. BASE
00013 1 04 00062  STA  GMD+1  SAVE MODE
00014 1 02 00054  LDA  GDL+C+1  LEADER COUNT
00015 1 04 00055  STA  GDLC+1  TO LEADER COUNTER
00016 1 02 00044  LDA  GDDR+1  FETCH DIRECTION XAC
00017 101040  SNZ  TEST FOR BIDIRECTIONAL
00018 0 01 00033  JMP  KB1  YES
00019 1 04 00045  STA  GD50+1  SAVE ADR W I.G. ON INTR
0001A -0 01 00336  KB2  JMP*  GT
0001B  EJCT

Slide 4.04  DDP 516 Instructions
16 bit word
8 bit offset

256 word page
512k word
store
4k peripheral
CSR's

E (normal state)

O L G
1 X X[P+G]
2 Y Y[G]
3 YI ind(Y[G])
4 YB Y[B+G]
5 ZB Z[B+G]
6 YW Y[W+G]
7 YWI ind(Y[W+G])

Slide 4.05 Modular One Addressing

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>loc [R]</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>loc [R] {R := R+2}</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>loc[loc[R] {R := R+2}]</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>{R := R-2} loc [R]</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>loc [ {R := R-2} loc [R]]</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>loc [ R+X]</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>loc [loc [ R+X]]</td>
<td></td>
</tr>
</tbody>
</table>

Slide 4.06 PDP-11 Addressing
Slide 4.07  PDP-11 Instructions
Slide 4.09  Input Character
Background process or SLIH is interrupted

Hardware interrupt → FLIH reply
              set lower interrupt resume

Background process is interrupted
or SLIH continues

Lower interrupt → SLIH scheduler reply
                  end: clear
                  interrupt section
                  resume
Simple Peripherals

- KBS
- TKS
- PRS
- LPS

Disks

- RCCS
- RFCS
- RKCS

Auxiliary registers for disks:

WC  BA  DA  ER  MN  DS

Slide 4.11 Control and Status Registers
Slide 4.12  Interactive Input Handler
Multi-process system has several LOGICALLY separate sequential processes.

(number may vary)

Multi-processor system has several PHYSICALLY separate sequential processes.

(number is fixed)

Slide 5.01 Processors and Processes
(each processor may perform several processes)

against

unauthorised

garbling
writing

spying
reading

stealing
execution

Segment: contiguous area of store,
uniform access rights

Base
Length (page quanta)
Access control

Slide 5.02 Protection
Read, write, execute
\{ General
Read, execute
\}
Read data
Write data
Execute
Read capability
Write capability
Enter capability
\} System 250
Instruction space
Data space
\} PDP-11/45
<table>
<thead>
<tr>
<th>Write</th>
<th>A</th>
<th>T</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>O</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>4</td>
<td>X</td>
</tr>
<tr>
<td>P</td>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

A : Abort before

T : Trap after

P : Permit

Separately for I-space & D-space

Slide 5.04   PDP-11/45 Access Modes
Slide 5.06  System 250 Multiprocessor System

Slide 5.07  Two-computer System
send 'ATTN'  (ATTN interrupt)
(Read interrupt) set Read (channel program)
send data    accept data

 Werk interrupt) set Write (channel program)
accept data    send data

80 byte basic message, 8 byte header
72 byte text
H: Hub computer
R: Rim computers
Slide 5.10  Protocol for text Message Transfer
DATA SOURCE

PROLOGUE

TEXT TRANSMISSION

EPILOGUE

DATA DESTINATION

ENQ

SOH

HEADER (8 BYTES)

STX

TEXT (VARIABLE LENGTH)

ETX

DELAY

LISTEN OUT FOR ERROR REPORT FROM SOURCE.

Slide 5.11  Basic Communication Program Structure
SYSTEM ASPECTS OF SMALL COMPUTERS IN PARTICLE PHYSICS:
A PERSONAL VIEW

B. Zacharov*)

CERN, Geneva, Switzerland

1. INTRODUCTION

The purpose of these lectures is to talk about small computers. Not about their detailed properties, neither about a comparison between different small computers, for that is the subject of another set of lectures, but about the way in which they are used in high-energy physics. And since small computers are hardly ever used in this field as a complete solution to some identifiable problem, this discussion can only take place in the much wider context of all the equipment and all the associated techniques that inevitably occur. Thus, what we shall be talking about is the system aspect, namely the role of the small computer as one component in some total system necessary to solve a given problem. Often the total system is quite complex, so it will not be possible to discuss all the pertinent details. Nevertheless, I shall try to describe as much as possible to show why it is, in each case, that it has been necessary to use small computers.

The development of applications of small computers in high-energy physics has been extremely rapid. In addition, the properties of available computers have evolved, and are continuing to evolve, at a very great rate. Finally, mainly as a consequence of mass integration, the cost and the reliability of any given system has improved in the last years in a way that can only be described as phenomenal. Therefore, it is most important to remember that what I shall say can only represent the situation as it is now, and

*) On leave from the Science Research Council, Daresbury Nuclear Physics Laboratory.
that, almost certainly, this situation will have radically changed
within the next five years. As an illustration of the sort of evolu-
tion that is still taking place, I can mention that in 1960 at
CERN there was only one single digital computer, a Ferranti MERCURY,
whose purpose was to serve the computational needs of all the exper-
iments, the theoreticians and also accelerator operation and dev-
elopment; it was considered to be a large computer. In 1963, the
same machine was already monopolized by a single counter group and
regarded as a small computer. By 1965, no one really wanted to use
it at all, so inferior were its characteristics compared with other
available small computers.

In addition to the very rapid rate of evolution of small com-
puters and their applications, there is also the problem that not
everyone agrees about the role of such computers in high-energy
physics - although, of course, everyone agrees that they are useful!
Therefore, whatever I shall say will necessarily reflect my personal
view. In particular, it is my view that the role of the small com-
puter will soon become rather like that of the oscilloscope, namely
that it is an indispensable and universal tool, used in all branches
of high-energy physics, but that it will not be a particularly exciting
ing thing to talk about. My feeling is that this is about the last
occasion when anyone will consider it worthwhile to single out small
computers as an isolated topic in the instrumentation of high-energy
physics, although the system aspects will remain interesting.

Small computers have played a decisive role in the past in the
development of high-energy physics. In the future, I feel that they
will just be another component of a complex data-processing system,
generally terminating in powerful central computing facilities.
What I shall try to emphasize, in describing my view of the present
situation, is this essential transition.

I shall divide the following into broadly two sections: the
first will be concerned with a general review of those areas where
small computers are used in the whole field of elementary particle
physics; the second will be about some detailed considerations of
some particular aspects, mainly interfacing of equipment and com-
munications links. I have not included accelerator control among the topics because of the lack of space. However, the process-control aspect of the use of small computers has been rather neglected in the past in high-energy physics, and nowhere in this field is it more important than in particle transport and acceleration.

2. **SOME GENERAL QUESTIONS**

Before proceeding to a survey of applications, it would be as well perhaps to try to define what is meant by the term 'small computer' and then to justify why anyone should use such a computer rather than a 'large' computer or, more precisely, a certain fraction of a large computer.

So what is a small computer? I think it is impossible to define, although, in any given example, it is usually possible to decide whether or not a given computer is small. A cynical view is to say that 'small' is an adjective used by anyone trying to justify the expenditure of funds on a computer in his particular case. But perhaps the best way is to list the principal properties and limitations of current small computers.

i) Word length generally ≤ 24 bits; up to 32 bits exceptionally.

ii) No standard hardware floating-point operations.

iii) Generally no hardware decimal or digit operations.

iv) Priority interrupt handling with direct-memory access or data break.

v) No concurrency.

vi) Rudimentary error checking and recovery.

Thus, it is clear that the PDP-11, PDP-15, DDP 516, IBM 1800, for example, are all small computers, although the PDP-10 probably represents a limiting case.

The performance of such machines is very impressive for all the hardware implemented features. Typical operation times are shown in Table 1, from which it can be seen that several arithmetic opera-
tions and the core cycle times are comparable to those of much more powerful computers. However, it must be borne in mind that the word length in the small computers is generally quite short and, partly because of this, the maximum size of core store is rather limited, seldom being greater than 32K words.

<table>
<thead>
<tr>
<th>Processor Type</th>
<th>Core Cycle Time (μs)</th>
<th>Word Length (bits)</th>
<th>Integer Operations (μs)</th>
<th>Load (μs)</th>
<th>Store (μs)</th>
<th>Floating Operations (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDP-8</td>
<td>1.5</td>
<td>12</td>
<td>3</td>
<td>6**</td>
<td>6.5**</td>
<td>3</td>
</tr>
<tr>
<td>PDP-15</td>
<td>0.8</td>
<td>18</td>
<td>1.6</td>
<td>7</td>
<td>7.3</td>
<td>1.6</td>
</tr>
<tr>
<td>MODULAR I</td>
<td>0.75</td>
<td>16</td>
<td>1.5</td>
<td>2.1</td>
<td>4.6</td>
<td>1.5</td>
</tr>
<tr>
<td>HP216</td>
<td>1.6</td>
<td>16</td>
<td>3.2</td>
<td>19.2</td>
<td>20.6</td>
<td>3.2</td>
</tr>
<tr>
<td>DDP 516</td>
<td>0.95</td>
<td>16</td>
<td>1.9</td>
<td>5.3</td>
<td>10.6</td>
<td>1.9</td>
</tr>
<tr>
<td>360/65</td>
<td>0.75</td>
<td>32</td>
<td>0.65</td>
<td>4.45</td>
<td>8.45</td>
<td>0.65</td>
</tr>
<tr>
<td>CDC 6600</td>
<td>1</td>
<td>60</td>
<td>0.3</td>
<td>1.0</td>
<td>-</td>
<td>0.8</td>
</tr>
</tbody>
</table>

*) subroutine only
**) average time

Table 1 Typical operation times for different processors

The cost of small machines can also vary considerably, ranging over nearly two orders of magnitude from about £2.5 K. At the minimum, the cost will be for a small digital controller, with no peripherals except a paper tape reader and punch, and only 4 K words of main memory; at the maximum the cost will include several direct-access storage peripherals, magnetic-tape transports and 32 K of core store.
In a general way, nearly all the features possessed by small computers are also to be found in larger machines. Indeed, they are usually restricted versions, and large machines can nearly always be found to perform better for any given operation. There is, however, one area where small machines usually do better, and that is in parallel-data access and priority interrupt schemes. Most large machines have ports between central memory and the outside world. However, access to these ports is generally only through very complicated data channels and, even then, the transfers can only be made through a very complex protocol. So, even though many large computers have very sophisticated memory organization, with interleaving and even error correction, and also the transfer rates in the channels can be up to many megabytes/s, the time taken to establish a connection and transfer the first word, byte or character can be as great as several milliseconds. The main reason for this is, of course, understandable, and it is due to the difficulty not only of interrupting any task currently executing under a complex multiprogramming operating system, but also in identifying and queueing the source or sink of the external data. For these reasons also, external interrupts cannot be handled in the very simple way that is adopted in small computers, far less can there be the comparable priority interrupt schemes. Of course, there are priority interrupt sequences in large computers, for example in the IBM system 360. However, the interrupts there can be not only from external sources but also from several different internal ones, both hardware and software.

Some large computers, for example the CDC 6600, do not even have the possibility of external interrupts, and the best way that external data sources (or sinks for that matter) can be handled is by status flags which are periodically examined by the peripheral processors. It can take quite a time therefore, in such systems, for data transfers between central memory and external devices to be initiated.

By contrast, the problem of transferring data, for example into the core store of a small computer with data-break facilities, is
almost trivial. All that is necessary, at the end of execution of a current operation after an interrupt, is to store the contents of the memory address register (and perhaps certain other registers), load it with a new address appropriate to the transfer of parallel external data, and then to perform the required data transfer either into or out of memory. The memory address register contents are then restored and the original program can continue execution; the whole data-break period is seldom greater than three memory cycles and frequently it requires only a single cycle.

Nevertheless, there are features which can be added to large computers which offer facilities very similar to data-break access on small computers. For example, in the IBM 360 system there is a peripheral parallel-data adapter, the IBM 2701, that can be attached to one or more channels, which gives access for up to four fast parallel-data input or output registers. Transfers of a single word to or from a buffer register can be initiated quite rapidly following a suitable interrupt.

Thus the question arises why it is that small computers are used at all rather than large ones.

Perhaps the answer is partly an economical one for, if the requirement is for integer manipulation and arithmetic only on small words, then we can see from Table 1 that certain small computers have a performance which is comparable with much larger and very much more expensive processors. However, the main reasons that small computers have been used are partly historical and partly due to the not always rational desire to remain independent of other computer users. In high-energy physics, small computers were used first of all to overcome the difficulties of data acquisition that arose with the advent, first, of scintillation counter hodoscopes and, later, with the innovation of sonic spark chambers. Up to that time, the principal limitations in computation had been in post-acquisition analysis, where the data had been recorded on film either from bubble chambers or from optical spark chambers. The introduction of devices that could register interaction events yielding data rates of $10^5$ bits/s, or even more, completely changed the nature of counter
physics and, although there were some attempts to solve this problem by scaler systems whose contents were then read-out onto magnetic tape, it was soon realized that the small computer as an interruptable buffer device was ideally suited for the role. At that period (1963-1966) the central computers at high-energy analysis centres were just not capable of solving the data acquisition problem of even one counter experiment, far less that of the many concurrent experiments around any given accelerator.

Once small computers had been introduced as standard features of counter experiments, the range of their application became greatly extended. Also, as wire chambers, with various types of read-out system such as cores or magneto-strictive lines, were developed, small computers became not merely useful as data-acquisition devices but indispensable. Additional tasks, such as sample event monitoring, instrumentation checking and data display, were introduced to the repertoire of what small computers were expected to do - but at a price. With each new task, the size (and cost!) of the small computer became not so small. We are still in this period of expansionism, and some counter groups have extended their small computer to the extent that they can perform on it a sizeable fraction of their complete event analysis. The negative aspect of this situation is that, in the past few years, large computers have also developed, and they can now do many things more efficiently and economically that small computers are being asked to do. It seems to me that there is now only one aspect of counter physics data processing that can be better performed by small computers and that is demand-handling and fast data acquisition. Thus, far from extending still further the 'small' computer facilities local to any experiment, their size should be reduced to the absolute minimum consistent with buffering input data and presenting output to the users; all the other tasks should be relegated to the large machine. However, this can only be done properly by high-speed data links, and I shall have more to say about this later.

I have indicated how small computers have come into widespread use in counter physics, to the extent that modern counter experi-
ments could not be implemented without them. However there is another broad class of applications where small computers have been used with considerable success, and that is for dedicated computational tasks, where the total load is within the capability of a small computer. Typical of such problems is the calibration of magnetic elements such as bending magnets or quadrupoles. In these applications, the task is a repetitive one where the computer is necessary to automate the field measuring and recording process. The task is usually considered to be complete when all the relevant data have been recorded.

The argument that was generally made when defending the use of a small computer in some automatic measurement or control application was that it is much too complicated to solve the problem on a time-shared multiprogramming computer and, in any case, it is more economical and more reliable to use a dedicated small computer. At a certain historical stage this was correct. However there are now very few examples in high-energy physics institutes (where there is always a large central computer) where the argument has any force. The point is that nearly every application that starts out as a simple measuring or control application, escalates in the computational requirement; a visual display is required, or the data must be recorded on magnetic tape, or there must be some partial data processing, and so forth. As soon as these additional requirements are added, it nearly always becomes much more effective to use a direct link to a large computer.
3. **A SURVEY OF APPLICATIONS**

I have put forward some radical ideas which indicate my view that the role of the small computer in most high-energy physics applications should be a subordinate one, as one intermediate device in the data processing chain terminating in a large computer. Nevertheless, before considering that point of view further, it is useful just to review the applications of small computers as they are at present.

3.1 **Data acquisition**

The data acquisition problem arises principally in the area of counter experiments. There are also some problems in process-control applications, where it is desired to monitor a set of analogue or digital parameters, but I will discuss that separately. The question of bubble and streamer chamber data acquisition does not really involve small computers to any significant extent, so I will not consider that topic further; the situation may change in the future with the development of high-resolution, sensitive signal-generating tubes.

With few exceptions, counter experiments register events either by a counter hodoscope arrangement or by some system of chambers operating in the spark or proportional region. Depending upon the nature of the interaction studied, both the number of bits needed to define the event and the rate of triggering can vary, as can the number of spurious events or background. Nevertheless there is a certain reciprocity; generally, the more complex the event, the less frequent the trigger - and vice versa. Similarly there is some compensation between the spatial and time resolution of the detecting system. Thus, the fastest detector (that with the shortest recovery or dead time), the scintillation counter, cannot be made to resolve easily a spatial element smaller than say 5 x 5 mm, although the recovery time can be made less than 20 ns. In these discussions it is important to note that time resolution was mentioned here in the loosest sense, and what is meant is not the time during which a given detector is sensitive to the passage of some particle or other, but the minimum time after a trigger that must elapse before
there can be another, completely independent trigger with 100 percent recording efficiency. Although both quantities are important, they differ usually by more than an order of magnitude. However, it is the longer time only, the recovery time, that is relevant to the data acquisition problem. In the case of a small scintillation counter, it is possible by very careful technique to achieve a true resolving time of 100 ps.

To give an idea of the range of parameters involved in data generation, Table 2 shows the range of recovery time, average and peak event rates and numbers of bits per event typical for the three principal detectors in counter physics. Two relevant quantities emerge from this table, the first is the aggregate data rate that might be encountered, and the second is the minimum time between successive events. In the present context, it is this second quantity that is most important, for it sets the limits to the data acquisition properties required of the small computer.

<table>
<thead>
<tr>
<th>Type of Detector</th>
<th>Typical Resolving Time</th>
<th>Typical Dead Time</th>
<th>Examples of Event Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scintillation Counter Hadoscope</td>
<td>1-2 ns</td>
<td>10-20 ns</td>
<td>$10^4$ bits/s, $2 \times 10^5$ bits/s, $2 \times 10^3$ bits/event</td>
</tr>
<tr>
<td>Spark Chamber System</td>
<td>1 µs</td>
<td>1 ns</td>
<td>$10^4$ bits/s, $2 \times 10^5$ bits/s, $5 \times 10^3$ bits/event</td>
</tr>
<tr>
<td>Proportional Chamber</td>
<td>50 ns</td>
<td>100 ns</td>
<td>$10^5$ bits/s, $4 \times 10^5$ bits/s, $2 \times 10^4$ bits/event</td>
</tr>
</tbody>
</table>

Table 2 Examples of Event Parameters for different detector systems in counter experiments.

Once having decided upon a computer for data acquisition it is unusual (and indeed quite expensive) to have more than a single external buffer; that is to say that there is only one set of hardware registers or scalers used in the experimental setup to store the bits which define a single event. Therefore, it is essential to empty this
external event buffer in good time for the next event to be recorded and, although this time is usually quite long on average, triggers can occur randomly and it is the minimum time between successive triggers that is important in order not to lose events and so greatly complicate the task of calculating the real rates at which interactions occur.

It can be seen that, for no triggers to be lost, it is not only the data-transfer rate that is important, since clearly the rate must be adequate to digest the total number of bits per event within the shortest time between two successive triggers, but also the interrupt-handling time must be much shorter than this period. Indeed, it is the sum of the interrupt-handling plus event data-transfer time that must be shorter than the minimum time between events, and it is precisely this fact that makes small computers much more suitable than large ones for data acquisition, since the data transfer rates are not all that different but the demand-handling times are.

There have been systems in which a fast data buffer is interposed between the event recording registers and the data acquisition computer. These buffers are usually magnetic core stores with suitable logic to handle the interrupts and data flow. However, the whole idea of a derandomizing buffer seems to reflect more the deficiency of small computers rather than a real system need, and, although there was a good reason to introduce such a component into the data-acquisition chains in the past, the performance of modern small computers is now so good that only extreme circumstances justify the introduction of intermediate buffers. Even then, it is usually adequate to have a single additional event buffer, since an appropriate statistical calculation will show that if the event rate is so fast that more than two external event buffers are needed to smooth out the data rate to the level that can be accepted by a good small computer, then the average data rate is likely to be so large that the total recording and processing load will be unmanageable.

To illustrate the above remarks, we can consider the case of a
small computer of 1 μs cycle time which stores events of 250 words (16-bit) each. Then, if each interrupt requires about 30 cycles to be handled before transfers can start and also requires 20 cycles for restoring, the fraction of events stored with an aggregate data rate of 200 events/s varies only between about 0.92 for one external event buffer to 0.99 for two or more. Clearly there is not much sense in providing here the additional external buffers. On the other hand, even if the average event rate were 2000 events/s, the fraction of events stored would be already about 0.68 for a single external buffer, and two external buffers would bring this fraction to 0.90. However, even though it could be argued that the extra buffer would be worthwhile, the aggregate data rate in this particular case would be 250 x 2000 x 2 = 10^6 bytes/s which is such a colossal data rate that it could barely be buffered even by the fastest disk stores (certainly not on magnetic tape) and I doubt that there is any processing system which would perform the required bulk processing (unless the total number of events were small, in which case there would hardly be any point in performing such fast data acquisition!!).

Thus, small computers certainly can and do solve the data acquisition problem for counter physics. They do this principally because the interrupt-handling time can be very short and, thereafter data words can be transferred (using direct memory access) at the memory cycle rate.

3.2 Buffering and packing

It can be seen from the foregoing discussions that it is not sufficient merely to solve the data acquisition problem, for the data have still to be transmitted to a large computer for subsequent analysis in all but very exceptional cases. The manner of transmission can be either by magnetic tape or by direct data link, but in either case there can still be the additional problem of buffering the data.

In the case of magnetic tape it is usual to write in some sort of standard BCD format, in order that there are no difficult problems of decoding binary patterns. This is not an exclusive rule,
but it is one which not only allows the data to be easily and compatibly transmitted to any other large computer, but it is reliable and not unduly wasteful, since the data tapes are invariably re-written and merged at the large computer in suitably compressed form for archive storage. Now it is essential for BCD recording to restructure the data in memory, if for no reason other than to minimize the frequency of inter-record gaps, and it is usual to pack the data of several events together into one logical record on tape. But, in any case, there are usually difficulties in writing magnetic tape on a small computer concurrently with other operations such as data input along a direct memory access channel. Therefore, it is well to minimize the magnetic tape output operation time and this, in turn, requires that the record length be as long as reasonably possible.

When a direct data link is used to transfer data from the small data-acquisition computer to a large one, problems also occur if the events are transferred singly. This is because the time taken to establish the connection along a link is usually quite long, and often comparable or even longer than the time taken actually to transfer the data. There are significant advantages, both from the point of view of the small computer and from that of the large system, to transfer data in large blocks, because of the long interrupt time on the large computer, because of the fact that block transfers can occur at channel or even core rates, and because of the need to minimize the amount of time the small computer spends on tasks other than data input and demand handling.

It can be seen that because of the requirements of transferring data from the data-acquisition computer, it is necessary to buffer the data. For these reasons, it is also usual to compress the data somewhat by packing as much information into data words as possible. Thus each small computer is generally used to buffer the incoming data as well as for data input. The extent of data buffering is obviously dictated by the actual average and peak events rates and also the input characteristics of the storage or link channel. There is also the important factor that data is generated in most
counter experiments in definite bursts. For example particle beams are available from the CERN Proton Synchrotron usually during about 300 ms every 3 seconds or so, while at DNPL, the electron synchrotron gives about 1 ms of particles every 20 ms. In either case, several events can be generated during each burst.

3.3 Sampling

One of the greatest advantages of having a computer on-line for data acquisition is that it is also available in principle to do a running check of the satisfactory performance of the experimental equipment. Broadly speaking there are two areas that need to be checked, the first is the functioning of the hardware, and the second is the test of the kind of interaction that is being recorded.

In general, the monitoring of experimental equipment is done by sampling a range of parameters, and then checking them against a standard table of stored values; if the new values are within a certain tolerance of the stored values, then it is assumed that the equipment is correctly functioning. The hardware monitoring is implemented in one of two ways, or a mixture of both. Either the on-line computer periodically executes a routine that scans a set of external registers and inputs the current contents of these registers, representing the required parameters in digital form, or there are external hardware devices that either periodically interrupt the computer and input the appropriate parameters, or they input them when something anomalous has occurred (such as the failure of a vacuum or a magnetic field, or there is a hydrogen leak!!). Of course, the input parameters may be analogue quantities, and it is quite common to have an analogue-to-digital converter that is multiplexed by a suitable matrix switch to scan a number of parameters of interest.

A more sophisticated equipment check that is sometimes used is to periodically trigger on charged beam particles and, in the case of a magnetic spectrometer experiment, to record the passage of a beam particle both with the analyzing magnet switched on and switched
off. Several powerful checks can then be made. For example, all the beam particles should have a profile at the target which matches the physical target cross-section. Also, the momentum distribution of the beam particles as measured by the analyzing magnet should be the same as that of the incident beam. With the analyzing magnet switched off, the beam particles should be travelling in straight lines. In all these tests, there is a check not only that all the equipment is correctly functioning, but also that there are no systematic changes that are slowly occurring.

The tests and monitoring that have just been outlined serve to check the hardware of the experimental apparatus. What the tests cannot do is to indicate whether or not the right kind of event is being recorded. Of course, it is very difficult to say what is the 'right kind' of event, even in specific circumstances, and it is absolutely impossible to generalize. However, for each experiment, there are usually a number of sample calculations that can be performed which help to check whether the physics of the experiment is correct or not. An obvious example here is to check a random sample of events and to plot the distribution of vertex positions. Clearly, if this distribution is not within the target limits, then there is something wrong.

The use of elastically scattered events in calibration and checking is a fairly widespread practice. First of all, in an experiment designed to record elastic scatterings, a random sample of events should actually show some evidence that the trigger criteria are correct. For example, the events should be co-planar, or there should be the appropriate balance of relativistic four-momenta before and after scattering. Even with a missing-mass spectrometer designed for a search of inelastic interactions, a very powerful test can be made with elastic events, when clearly the missing-mass distribution should not only be centred at the actual mass of the scattered particle, but the width of the distribution should be characteristic of the resolution of the whole apparatus.

Now I have mentioned how useful it is to be able to check the correct operation of experimental apparatus, and also to monitor the
kind of physical interactions that are taking place. Also I have indicated the nature of the sampling that it is possible to do. What I have not said however, is how realistic it is to perform all these checks on a small data-acquisition computer. Unfortunately, the short answer to that question is that it is not at all a good idea to expect a small computer to perform most of the tasks I have outlined, and there are several reasons for this. The first drawback is that most of the calculations that are useful, of the kind I have mentioned, require floating-point arithmetic, and it is most unusual (and expensive) to have hardware implemented floating-point operations available on a small computer. Secondly, all these calculations generally require loading the appropriate routines and tables from outside core, and that demands some sort of direct-access storage like drum or disk (or sometimes DECTAPE). The third problem is that the time taken to perform the appropriate sampling calculations is usually very significant on a small machine, so that, at the very least, there is a strong limit to the frequency of such checks. At the worst, sampling actually interferes with the output of data to magnetic tape or data-link, so that the amount of buffer space in the memory of the small computer has to be increased still further.

It must also be remembered that any interrupt to most small machines can generally only be handled at the end of execution of a current operation or when a higher priority program segment has been serviced, and this could introduce a delay of even hundreds of microseconds.

My conclusion is that the small computer should not be expected to perform all these tasks that can be executed so simply on a larger machine with direct-access facilities, hardware floating-point operations, multi-tasking and also, frequently, ATTACH or common task facilities. Then the small computer could continue to do what it does best, and the rest of the computational task could be performed much more expediently (and in a high-level language) on a linked larger computer. Of course, there may be some areas of the sampling process that can still be better performed on the small
computer, such as parameter sampling and comparison, for that is typically a demand-handling plus integer-arithmetic problem. However, that is generally a minor aspect of the monitoring and checking question.

3.4 Feedback of information

I have referred to sampling and checking and I have described some of the techniques used. What I have not said is how the results of these processes are conveyed back to the experimentalists or, sometimes, to the equipment itself. Of course, this problem is sometimes straightforward. For example, in the case of the dreaded hydrogen leak to which I have already referred, it is adequate in the first instance to cause a bell to ring, or a klaxon to sound, or an alarm light to flash - or all these things together. However, it is important to note that it is just as essential to know when things are correctly operating as it is to know when something is wrong. The psychological value of routine checks cannot be stressed too greatly. Thus it is principally to the human experimentalists themselves that information has to be fed back.

In counter experiments, and indeed in many other on-line systems, there are generally two ways of conveying information back to human-beings. The first technique is for the human-being to ask for a display of a known set of parameters, or the computer can arrange to present these displays periodically. In this case, it is a simple thing to do to pick out any anomalous circumstances. The second method is to periodically display a summary of up-dated distributions, generally in histogram or scatter diagram form.

In all these cases, even in the most trivial example, there is no doubt that the best way to actually communicate with the user is by using a graphical display. It is clear that the most natural method of conveying information about a set of many parameters is by means of a pattern, because human-beings are above all highly skilled in pattern recognition. Even in the relatively few instances when precise numerical information is needed, the required data can be displayed on a graphics terminal. There is then a very
powerful feature that can be used, namely that anomalous values of parameters can easily be indicated, for example by intensity modulation or "blinking".

To illustrate the point about visual display, we can recall the previous examples of monitoring both apparatus and physics by calculating the beam particle intersections within the target region and by calculating the event vertex positions. Clearly, the best way to communicate the relevant information to human-beings is by graphically displaying the appropriate vertexes or intersections, either in two orthogonal plane projections or in a two-dimensional representation of the three-dimensional distribution, with the outline of the target indicated. An example is shown in Fig. 1.

Because of the very great power of graphical display techniques in feeding back information to users, it is not surprising that nearly every counter experiment with an on-line computer is now equipped with at least one visual display, usually a direct-view storage tube terminal. But here, once again, difficulties arise because small computers are not well suited to graphics applications. The question here is not whether small computers are able to be used for these tasks, for they can be and are employed for this purpose. Rather it is that, in order to properly drive a graphics terminal, even without interactive facilities such as a light pen, it is necessary to extend the configuration of the small computer, especially in respect of central memory and also backing store.

Once again it is important to realize not only that all these facilities are readily available on any large computer to which the small computer can easily be linked, but also that the facilities are available to users in high-level languages and with proper data and program library facilities.

3.5 Process control

The use of small computers in industry for the control of definite processes is very well developed and has been known for at least ten years. It is surprising therefore that the high-energy physics community has been very slow in employing similar techniques for
Fig. 1  Distribution of Interaction Vertexes in Counter Experiment
monitoring and controlling systems of sufficient complexity to warrant an automated process. The situation is changing rapidly, however, and there are already several areas where small computers have been used with great effect to reduce drudgery or to improve reliability.

The most obvious area where there is a complicated process to control is in the particle acceleration system. The essential statement of the problem is that, first, the system parameters need to be monitored to check whether they are "normal" or not. What is meant by normal here will not be defined, because it depends upon the context of measurement; sometimes it is necessary to know whether parameters are changing or not, while on other occasions one needs to record the distribution of their values so that these same values may be restored at some later time; this aspect is known as data-logging. Second, it is necessary to be able to set certain parameters to a desired range of values. Finally, and most difficult, one wishes to control at least certain sub-systems to conform with a defined regime.

A very much simpler task than accelerator control is the complete control of one or more beam transport lines. Here the total number of elements and parameters is generally quite small, and the precise values of each of the parameters is known beforehand for each of the required transport regimes. Any changes in parameters are likely to occur either catastrophically or to be sufficiently slow that they can be restored by simple negative feedback. In either case, a small computer can be most useful to actually perform the total control process. All that is required is that the whole range of relevant parameters can be periodically scanned by digital or analogue multiplexing, and that a corresponding set of control parameters can be adjusted by a suitable set of output signals from the computer. For example, the control field of a magnetic element is periodically monitored and the corresponding magnet current (either the main current or auxiliary winding current) is controlled.

In beam transport control, it is clear that the problem of setting up the transport elements to a desired regime, in accordance
with a table of values, is relatively straightforward when using a computer, once a suitable procedure has been defined. Certainly, much laborious manual setting is eliminated in this way. Secondly, the question of checking to see that the total transport system is functioning correctly is also fairly simple, although the monitoring of beam profiles in the target regions of various experiments is sometimes a problem. The only difficult part of the whole system is to calculate what should be the values of certain parameters when a given beam is not in the correct position or it does not have the correct properties (momentum, convergence etc.). However, this problem is usually avoided by performing such calculations beforehand and then just storing the required table of values. In general, the statement can be made that small computers are more than adequate to perform the tasks associated with the first two aspects, but that only a large computer (with human guidance) is sufficient to solve the last.

The last example of process control that I wish to mention is a relatively recent innovation, which concerns the automation of certain routine tasks that inevitably occur in counter experiments. In every such experiment there are always control parameters that have to be set within a certain range, defined either by manufacturers specifications or by actual operational conditions. An example of the first might be the voltages on focussing electrodes in photomultipliers, while an example of the second could be the time delay in a coincidence channel in order to achieve the maximum coincidence counting rate.

In this aspect of process control there are two problems, the first is to set up the correct operating values of parameters in order to obtain a desired system characteristic, while the second is to monitor parameters and it may be necessary to re-adjust them to compensate for drifts. From an operational point of view, both aspects can be implemented by the same automated system using a small computer, although the first aspect is one which is more relevant during the setting-up stage of an experiment while the second is generally only relevant during the data-taking phase. One
example of a system designed to perform these tasks is the ACE system (an acronym for Automatic Calibration of Equipment), which provides for a monitor of several parameters of interest in counter experiments, such as photomultiplier EHT, coincidence counting rates, etc., and also possesses a standard reference counter channel in which all the parameters can be varied. By the use of light-emitting diodes in each scintillation counter, and by suitable control algorithms, it is possible not only to scan any required range of parameters, but also to determine their correct operating values. For example, it is easy to determine the plateau region of operation for any given scintillation counter and, in this way, to establish the correct value of EHT on the relevant photomultiplier.

The ACE system could be used, in principle, to actually set the correct values of a given set of parameters automatically. In practice, this is considered to be extravagant, since an automatically controlled hardware device for each parameter would be quite costly. Instead, what the ACE system does is to define the correct values of parameters and communicate the information to users by means of a graphical display. The actual values of the parameters are then set manually.

3.6 Multiplexing

Small computers are used within each given experiment to serve several purposes and to communicate with several data sources and sinks. In this sense they are already used as a multiplexing device. However, there is a much broader application of small computers as multiplexors and buffers, and that is to serve several experiments. This aspect of the use of small computers is becoming increasingly important as more and more local small computers are connected to central facilities, not only for common data processing and active storage but also for the feedback of information, particularly in graphical form.

Strictly speaking the application that is being discussed here is only one aspect of data links between local small computers and one or more central large computers. However, it is a distinct application, and it arises principally from the fact already mentioned,
that large computer systems are generally not well suited to handling direct data communications, far less are they matched to the task of handling many such links, each with two-way or duplex traffic.

It is true that there are hardware multiplexing devices for the attachment of several data links to large computers. For example, the IBM 2701 data adapter already mentioned can be used for up to four parallel-data paths. However, the problem is that these devices not only provide no buffering capability beyond a single register for each channel, but they have no way of coping with the more complex, but nevertheless real, requirements of priority demand handling or data conversion and compaction. What is needed functionally is a device that can handle all the incoming messages and data blocks without effectively retarding the traffic, queue and buffer the data and then transmit the data to the central facilities in suitably blocked form in a way that can easily be handled by the large system. This is an ideal task for a small process-control computer with priority interrupt and multiplexing facilities.

An example of the application we are considering is the use at DNPL of an IBM 1802 as a front-end buffer-multiplexor between up to eight fast duplex links and the central IBM 360/65. The 1802 acts as a switch, buffer and funnel. However, it is completely transparent to the users, who write their programs on the central computer to handle the transfer of messages or data either to or from their local small computers or terminal devices.

3.7 Interfacing and terminals

In high-energy physics it was usual, up to only a few years ago, to build self-contained digital instrumentation, not only with a separate autonomous read-out or control system, but also with separate checking facilities. Typical of this philosophy were scaler read-out systems, which not only allowed the scaler contents to be coded and written onto compatible magnetic tape, but also incorporated complex visual displays on decimal indicators such as Nixie tubes.

There is now a strong trend away from self-contained systems
towards systems of compatible interfaces between a small computer and experimental equipment. The small computer can then be used either independently or linked to other devices such as a larger computer.

The main requirements in modern digital instrumentation used in high-energy physics are modularity, compatibility and the ability to incorporate already developed sub-systems into new, more complex systems. There is also the more general requirement to be able to get the digits either to or from a processing device (usually a large computer) as soon and as reliably as possible. It turns out that all those requirements can be very closely matched by the properties of small computers together with a standard interface. Many small computers exist which are suitable, but it would have been most uneconomical to introduce a computer to eliminate the inflexibility of special purpose hardware systems, while at the same time having to develop a new interface for each new type of computer or each new instrumentation system. For this reason a standard interface has been adopted, the CAMAC system. This system will be treated in some detail in the next section, but it is essential to realize that CAMAC is not only a system of interfacing between modular hardware and small computers, but also it is a convention for the signal and information dialogue that must pass between the two. The main advantage of the CAMAC system is that it enables hardware and software developed for one purpose to be smoothly incorporated into another system. And it is not only in the interfacing of experimental instrumentation that CAMAC plus small computers have found their use, but also in connecting remote terminals of all kinds, such as alphanumeric and graphics terminals. Of course, CAMAC is not the only modular interface system. However, it is quite evident that it is now the only system that has really achieved international recognition, and it is the only one which we shall consider further.
4. THE ROLE OF CAMAC

In the last section we have surveyed some of the many different ways in which small computers have been used, and are still being used, in high-energy physics. Out of this survey emerge certain systematic tendencies which I now wish to consider in somewhat more detail. There are two broad aspects that I shall discuss. First of all there is the whole question of CAMAC, which is undoubtedly having an extremely important impact not only upon hardware but also upon the shape of modern instrumentation systems. Secondly, I shall talk about data links, because of the importance they have in extending the power of central computing systems to remote users; as we shall see, the small computer is an indispensable component of link systems.

4.1 What is CAMAC?

The CAMAC system was developed to standardize the method for the transmission of data and control between instrumentation modules and a digital controller. In addition, it defines the precise mechanical format of both module and crate, together with certain signal and power supply conventions. However, there is nothing in the CAMAC specifications\(^2,3\) which defines what the digital controller should be; it could be a purely hardware device within a crate, or outside it, or it could be a computer. In this discussion we will restrict ourselves only to those systems in which a small computer is the controller. Moreover, the small computer will be used to drive a CAMAC system of crates in a branch configuration, as shown in Fig. 2. In principle, it is possible to use the computer to control each crate separately, using the crate controller in each case as an interface between the computer I/O bus and the CAMAC dataway in each crate. However, the CAMAC system has defined conventions for interconnecting up to seven crates in a branch, with only a single interface (the branch driver) between the computer and each of the crate controllers, and it is this system we will discuss.

A formal description of the CAMAC system is given elsewhere, but it is useful just to review the principal features here.
Fig. 2 CAMAC branch: Chain configuration
4.2 The Dataway

CAMAC modules, which are interchangeable, can occupy any one or more of 24 stations in a crate. Thus, there is a minimum module space, although modules frequently occupy multiples of two, three or even four spaces. The crate is just a structure for mounting the modules, supplying them with power, and providing an interconnection highway at the rear for data and control signals; this highway is called the Dataway. Fig. 3 shows a typical crate with 25 stations.

The physical Dataway is a wiring structure that interconnects the edge connectors, each with 86 pins, at every one of the 25 stations. However, the wiring is not a simple 86-way parallel bus, neither does the dataway extend simply to the end station, position 25. All control and data lines, except for two special ones, form a bus making these signals available at all the stations 1-24. The exceptions are N-lines, which select each station separately, and L-lines which signal an attention status for each station separately; there are separate N and L lines for each station, and these lines connect each station to the connector of position 25. Thus, position 25 has the special property that it is the only station that can select or address the other 24 stations. For this reason it is occupied by a special module, called a Crate Controller which, because it must have access as well to all the data and control lines of the Dataway, also occupies station 24. In the standard configuration which we are considering, with a single CAMAC branch of up to seven crates, each crate will have a standard double-width Crate Controller (CC), so actually there are only 23 stations then available for modules other than CC.

Table 3 lists all the signals that can pass along the Dataway, but does not give details of power lines, reserved lines and patch wiring. The main points to notice are that data can flow in both directions with words of up to 24 bits, but that commands and address information come only from the Crate Controller. Status information is transmitted from individual modules either in response to special signals from the CC or, spontaneously, on the L-lines. It can be seen that address information is in a mixture of codes, the module being addressed by an individual N-line while up to 16 sub-units within a module are addressed by a four-bit binary code on the A-
Fig. 3 CAMAC crate with single-width module
lines. The reasons for this are mainly historic, but the adopted convention does at least permit a simple way of concurrently addressing more than one module; on the other hand the module is addressed by the station number where it is located, so positional independence has been sacrificed.

<table>
<thead>
<tr>
<th>Title</th>
<th>Designation</th>
<th>Pins</th>
<th>Use at a Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>N</td>
<td>1</td>
<td>Selects the module (Individual line from control station)</td>
</tr>
<tr>
<td>Station Number</td>
<td>M1, 2, 4, 6</td>
<td>4</td>
<td>Selects a section of the module</td>
</tr>
<tr>
<td>Sub-address</td>
<td>M2, 2, 4, 8, 15</td>
<td>5</td>
<td>Defines the function to be performed in the module</td>
</tr>
<tr>
<td>Function</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Timing**

| Strobe 1 | S1 | 1 | Controls first phase of operation (Dataway signals must not change)               |
| Strobe 2 | S2 | 1 | Controls second phase (Dataway signals may change)                                 |

**Data**

| Write         | M1 - M24 | 24 | Bring information to the module                                                 |
| Read          | M1 - M30 | 24 | Take information from the module                                                |

**Status**

| Look-at-Me | L | 1 | Indicates request for service (Individual line to control station)              |
| Response    | Q | 1 | Indicates status of feature selected by command                                 |
| Busy        | B | 1 | Indicates that a Dataway operation is in progress                                |
| Command Accepted | X | 1 | Indicates that module is able to perform action required by the command        |

**Common Controls**

| Initialise | Z | 1 | Sets module to defined state (Accompanied by S2 and B)                          |
| Inhibit    | I | 1 | Disables features for duration of signal                                         |
| Clear      | C | 1 | Clears registers (Accompanied by S2 and B)                                      |

Table 3
Principal CAMAC Dataway lines
4.2.1 Dataway operations

Operations along the dataway largely occur in response to a NAF combination of signals, where a location A in a given module N executes some function F. Such operations are issued by CC and conform to a quite precise set of cyclic timing rules which we will not go into here. Now the NAF commands are directed to specific locations and there are clearly many possibilities. In particular, there are 32 choices for the 5-bit binary coded F-signals. However, most of these F-combinations are not free, as can be seen from Table 4. There are broadly three categories of function:

a) Control or status functions, not involving data transfers; for these functions F8 = 1

b) Read functions transferring data from modules to CC,
   F16 = F8 = 0

c) Write functions, transferring data from CC to modules;
   F16 = 1, F8 = 0.

In Table 4 there is a reference to Group 1 and Group 2 registers and, although there is no rigorous distinction between the two, the first group are generally taken to be data registers while the second are for storage or control purposes within the modules. Overwriting is an operation equivalent to 'clear and load' in computer terminology. Selective set and selective clear are analogous operations, but with operation only upon the bits selected.

In addition to the addressed NAF-commands, there are also the (C), (Z), and (I) lines, which are carried to all modules concurrently by separate bus lines. The initialize line (Z) is used to set all registers and triggers in modules into a defined state, usually the logical state that enables modules to be receptive to initial command sequences. The C-line is not exactly defined, but CAMAC usage is to employ it to clear all data registers. The inhibit line (I) can be used in any way the module designer wishes.

An important CAMAC rule is that both the (C) and (Z) commands must be accompanied by the strobe signal S2 and, since S2 can only be generated inside a crate by CC, it is impossible for any module
itself to clear or initialize the system. On the other hand the
I-condition, like the L-signals, can occur at any time, even outside
the CAMAC cycle. It is very important therefore that (I) be used
with great care, since it is unaddressed and available to all modules
simultaneously; in practice it is overwhelming usage to generate
(I) from CC only.

4.2.2 Status signals

There are four basic signals which indicate the status of any
given module. The first of these (Q) may be given by a module in
response to a dataway operation, and it can be given only by the
addressed module. (Q) is used in CAMAC to indicate not only that a
given feature of a module is actually present at the proper station,
but also to give the status of any feature selected by a NAF command.
Another use of (Q) is to respond to a 'status-request' command such
as F(8) or F(27) in Table 4.

It is clear that apart from routine checking, (Q) can also be
used as an interrupt 'flag', and it is indeed used in this way.
However, a much more direct form of interrupt can be achieved by the
(L) line. This line runs from each separate station to CC and it
can signal that it requires attention at any time. In this way, CC
can know at any time which of the 23 stations need attention. Now
the CAMAC specification does not say what (L) should mean, far less
what response there should be from CC; that is up to the equipment
designer. However, there is already one general problem that can
occur due to the fact that, not only can each sub-address in a mod-
ule generate an interrupt, but there is nothing to prevent it gener-
ating more than one interrupt. There are several ways in which this
problem is generally resolved, usually either by sequentially scan-
ning all sub-addresses in the module that has generated the (L)
signal, and using the F(8) command to generate an appropriate (Q)
response (Q = 1 for the sub-address with L = 1), or to read-out the
L-pattern onto the dataway using two special sub-addresses A(12) or
A(14) and the F(1) command. In the first case, it is clear that
there cannot be more (L) signals than sub-addresses, namely 16,
while in the second case there can be (n x 24) signals, where n is
<table>
<thead>
<tr>
<th>Code (F)</th>
<th>Function</th>
<th>Use of R and W lines</th>
<th>Function Signals</th>
<th>Code (F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Read Group 1 Register</td>
<td>Functions using the R lines</td>
<td>0 0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Read Group 2 Register</td>
<td>Additional functions using the R lines</td>
<td>0 0 1 0 1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Read and Clear Group 1 Register</td>
<td>Functions not using the R or W lines</td>
<td>0 0 1 0 1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Read Complement of Group 1 Register</td>
<td>Additional functions not using the R or W lines</td>
<td>0 0 1 0 1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Non-standard</td>
<td></td>
<td>0 0 1 0 0</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
<td></td>
<td>0 0 1 0 1</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>Non-standard</td>
<td></td>
<td>0 0 1 1 0</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
<td></td>
<td>0 0 1 1 1</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>Test Lock-at-W</td>
<td></td>
<td>0 1 0 0 0</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>Clear Group 1 Register</td>
<td>Functions not using the R or W lines</td>
<td>0 1 0 1 0</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>Clear Lock-at-W</td>
<td></td>
<td>0 1 0 1 1</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>Clear Group 2 Register</td>
<td>Additional functions not using the R or W lines</td>
<td>0 1 1 0 1</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>Non-standard</td>
<td></td>
<td>0 1 1 1 0</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
<td></td>
<td>0 1 1 1 1</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>Non-standard</td>
<td></td>
<td>0 1 1 1 0</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>Reserved</td>
<td></td>
<td>0 1 1 1 1</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>Overwrite Group 1 Register</td>
<td>Functions using the W lines</td>
<td>1 0 0 0 0</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>Overwrite Group 2 Register</td>
<td></td>
<td>1 0 0 0 1</td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td>Selective Set Group 1 Register</td>
<td></td>
<td>1 0 0 1 0</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>Selective Set Group 2 Register</td>
<td></td>
<td>1 0 0 1 1</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>Non-standard</td>
<td></td>
<td>1 0 1 0 0</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>Selective Clear Group 1 Register</td>
<td>Additional functions using the W lines</td>
<td>1 0 1 0 1</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>Non-standard</td>
<td></td>
<td>1 0 1 1 0</td>
<td>22</td>
</tr>
<tr>
<td>23</td>
<td>Selective Clear Group 2 Register</td>
<td></td>
<td>1 0 1 1 1</td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td>Disable</td>
<td>Functions not using the R or W lines</td>
<td>1 1 0 0 0</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>Execute</td>
<td></td>
<td>1 1 0 0 1</td>
<td>25</td>
</tr>
<tr>
<td>26</td>
<td>Enable</td>
<td></td>
<td>1 1 0 1 0</td>
<td>26</td>
</tr>
<tr>
<td>27</td>
<td>Test Status</td>
<td></td>
<td>1 1 0 1 1</td>
<td>27</td>
</tr>
<tr>
<td>28</td>
<td>Non-standard</td>
<td></td>
<td>1 1 1 0 0</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>Reserved</td>
<td></td>
<td>1 1 1 0 1</td>
<td>29</td>
</tr>
<tr>
<td>30</td>
<td>Non-standard</td>
<td></td>
<td>1 1 1 1 0</td>
<td>30</td>
</tr>
<tr>
<td>31</td>
<td>Reserved</td>
<td></td>
<td>1 1 1 1 1</td>
<td>31</td>
</tr>
</tbody>
</table>

Table 4 CAMAC Function Codes
the number of spare sub-addresses. However, it would be most unusual in any real situation to have more than a few sources of interrupts within a given module. In the second method A(12) can contain a 24-bit (L) pattern, while A(14) is the raw pattern masked with the contents of A(13).

As an illustration of both methods of (L) identification, let us consider a module containing four scalers, some of which may have overflowed and generated an interrupt. This is shown in Fig. 4, where there are flip-flops FF1, FF2, FF3 and FF4 storing the overflow (L) signals for each of four scalers; these flip-flops form part of the LAM status register A(12). In addition, there are the masking flip-flops FF5 ... FF8 enabling or disabling each L-source separately, and FF9 to disable all the L-sources collectively. The status register can be cleared either as a whole by the (Z), N.F(11)A(12) or N.F(10)A(α) commands, or individual bits may be reset by F(10)A(i). Similarly, the LAM mask register A(13) can be cleared by (Z) or by N.F(11)A(13); individual bits of A(13) can be either disabled or cleared by N.F(24)A(i) or N.F(23)A(13) respectively. Selected bits of A(13) are enabled by F(26)A(i) or set by F(19)A(13). In both the commands using A(13) explicitly, the use of F(19) and F(23) require an appropriate bit-pattern derived from the (W)-lines. A point to mention here is that A(α) can be any spare sub-address, but that in LAM-handling it is usual to have α = 0.

Assuming that all appropriate L-sources have been enabled and that FF9 is set (using F(26)A(α)), then, following generation of an L-signal because one of the scalers has overflowed, the precise source address can be identified by the sequential scan N.A(i)F(8) for i = 1, 2, 3, 4, or by a parallel read-out onto the Dataway using N.A(14)F(1). In the first case the L-source is identified at the station and sub-address for which Q = 1, so that the search takes a maximum of four dataway cycles; in the second method, a single cycle is needed to read-out the whole L-pattern. In either method it is usual to identify the source station address N by the sequential scan N.F(8)A(α) until Q = 1.

There is one other dataway line, designated (X), which was
Fig. 4 Some possible methods of (L) source identification

N.B. \( A(\cdot) \) is any spare sub-address, usually \( \cdot = 0 \)

\( S(\cdot) = \text{SCALER}(\cdot) \) overflow

\( R(\cdot) = \text{N.F}(\cdot) \) \( A(\cdot) \cdot A(\cdot) \) or \( \text{N.F}(10)A(12) \)

\( X(\cdot) = \text{N.F}(26) \) \( A(\cdot) + \text{N.F}(10)A(13) \)

\( Y(\cdot) = \text{N.F}(24) \) \( A(\cdot) + \text{N.F}(23)A(\cdot)B \) or \( \text{N.F}(10)A(12) \)
reserved until recently but is now incorporated into the CAMAC system. Its function is to transmit from a module the acceptance of a command. What is meant by 'acceptance' here is that X = 1 must be generated if the module is able to perform the command it receives, and one important use is by the system software to check whether or not modules (or even sets of modules) are present and correctly operating. In a complex system, where certain sections are already operative while others are still in the stage of debugging, the (X) response can be selectively masked.

It is clear that the function of the (X) line is generally quite different from that of (Q) which is intended to be used mainly for specific applications, for example demand handling and for block transfers.

4.2.3 Block Transfers

One of the most important applications of CAMAC so far has been in the transfers of blocks of data either from one register or from a group of modules, usually scalers or ADC's. CAMAC provides a convention, using the (Q) response, to facilitate such transfers either in the 'Repeat' mode, the 'Stop' mode or the 'Address Scan' mode. In the first mode, transfers are from a single register (sub-address) and Q = 1 each time the register is ready to transfer a data word; in the second mode, Q = 0 immediately after the end of a block is reached. For the address-scan mode, the principal CAMAC rule that is relevant here is that sub-addresses in modules used for the transfer must be sequentially assigned starting with A(0), for which Q = 1 during a read or write operation, and that Q = 0 for the first sub-address following.

To illustrate the address-scan procedure, Fig. 5 shows a suitable algorithm for transfer of a data block from a set of modules in a crate. It is important to note that the algorithm can be implemented by building special hardware, for example in the branch driver, or by incorporating appropriate features in a CAMAC control program.
Fig. 5 An algorithm for block transfer in address scan mode
4.3 Multi-crate Systems

So far, we have considered only the question of a single CAMAC crate, and not the problem of a multi-crate configuration. We shall now discuss the whole question of how to control a CAMAC configuration, where there is in general more than one crate.

As mentioned earlier, it is possible to control a CAMAC crate by means of a computer, using a special module in station 25 (or stations 24 + 25) which performs the dual function of a crate controller and also computer-CAMAC interface. In such a scheme, a CAMAC crate can be treated as if it were a peripheral device attached to the I/O channel of the computer. This approach has been adopted by a number of groups who claim the advantage that a CAMAC crate can then be addressed like any other peripheral device on the I/O bus and that suitable hardware can be incorporated in the crate controller to determine the complete address of an interrupt source, if necessary with priority included. In going now to many crates, the appropriate configuration can be termed 'radial' since each CAMAC crate is connected separately to the computer. Thus, there must be at least as many ports to the computer as there are CAMAC crates connected; on the other hand, the only limitation to the number of such crates is that set by the number of external device addresses.

An important disadvantage of a radial scheme is that there needs to be a separate, computer-dependent controller for each crate, and this tends to be most uneconomical; one of the greatest assets of the CAMAC convention is that it offers complete interchangeability of modules from one computer system to another, and this asset is wasted if, in doing so, it becomes necessary to change also the crate controllers.

The alternative to the radial crate configuration is a branch configuration, in which a number of CAMAC crates are interconnected by means of one single command and data highway, termed a Branch Highway. There is then a single connection between the computer and the branch highway, performed by an interface device called a Branch Driver. Clearly, in a branch configuration, when moving from one
computer to another, it is only necessary to change the branch
driver; moreover, since the branch highway protocol and signal
standards can be defined, the interconnections between all CAMAC
crates and the branch highway can be made standard also, and this
can be by a single standardized crate controller.

In the ESONE CAMAC convention, it is the branch configuration
which has been taken as the standard method of interconnecting many
CAMAC crates, and a specification has been written\(^3\) which defines
both the branch highway and a standard interconnection between
crates and a port on the branch highway, termed a CAMAC Crate Con-
troller. The CAMAC specifications also describe one particular
crate controller, termed Type A. We shall now consider the branch
configuration in more detail but, in doing so, we must not forget that
it is only one of many different configurations which are possible;
nevertheless it is the one which appears to offer the greatest
degree of interchangeability, and it is the only system endorsed by
the ESONE Committee.

4.3.1 The Branch Highway

Physically, the branch highway is of 66 signal lines, each of
which being actually a twisted pair, so that there are a total of
132 wires. Unlike the dataway, which is a fixed structure, there is
some freedom in the branch highway configuration. For example,
rather than the configuration shown in Fig. 2, where the branch
driver is at one end of the branch highway, it is also possible to
place it at some convenient point along the branch as shown in Fig.
6. Nevertheless, there are several parameters that are fixed, the
main one being that there can only be up to seven crates connected
to the highway; all connections to the branch highway are through
standard ports, each with a standard 132-way connector.

The branch highway is a parallel transmission line of between
70 and 100 ohm impedance, to which all signal sources are connected
by means of an intrinsic or wired-OR type of connection. In order
to provide the correct logic signal levels with this arrangement,
and in order to minimize spurious reflections, it is necessary to
terminate the lines in the correct impedance, supplying the correct
Fig. 6 CAMAC branch: Example of an alternative configuration
standing current (the so-called pull-up current). It is preferable actually to have a termination at both ends of the branch highway in order to attain the best signal characteristics such as speed and pulse shape, and the way this can be done for example for the data lines, is shown in Fig. 7. The limitation of number of crates to seven comes about from practical considerations. On the one hand, with modern integrated circuits, it is difficult to supply much more current than that required by all the highway terminations and signal sinks together; on the other hand, the total number of highway lines is limited by the availability of compact and reliable multipole connectors, and it is already difficult to find lines for all the required signals without increasing the number of crates beyond seven. However, it would be a very large system indeed that would fill all the $7 \times 23 = 161$ available stations.

![Termination scheme for branch highway lines](image)

**Fig. 7** Termination scheme for branch highway lines

The branch highway operates in an analogous way to the CAMAC dataway. However there are some important differences both in the
<table>
<thead>
<tr>
<th>Title</th>
<th>Designation</th>
<th>Generated by</th>
<th>Signal Lines</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crate Address</td>
<td>BC01 - BC07</td>
<td>Branch Driver</td>
<td>7</td>
<td>Each line addresses one crate in the branch</td>
</tr>
<tr>
<td>Station Number</td>
<td>BN1, 2, 4, 6, 15</td>
<td>&quot; &quot;</td>
<td>5</td>
<td>Binary coded station number</td>
</tr>
<tr>
<td>Sub-address</td>
<td>BN1, 2, 4, 6</td>
<td>&quot; &quot;</td>
<td>4</td>
<td>As on Datavay A lines</td>
</tr>
<tr>
<td>Function</td>
<td>NF1, 2, 4, 6, 15</td>
<td>&quot; &quot;</td>
<td>5</td>
<td>As on Datavay F lines</td>
</tr>
<tr>
<td>Data</td>
<td>Read/Write</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BRMA - B8224</td>
<td>Branch Driver (A) or</td>
<td>24</td>
<td>For Read data, Write data, and Graded-L</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Crate Controller (R, SL)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td>Response</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Q</td>
<td>Crate Controller</td>
<td>1</td>
<td>As on Datavay A line</td>
</tr>
<tr>
<td>Command Accepted</td>
<td>EX</td>
<td>Crate Controller</td>
<td>1</td>
<td>As on Datavay X line</td>
</tr>
<tr>
<td>Timing</td>
<td>Timing A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BA</td>
<td>Branch Driver</td>
<td>1</td>
<td>Indicates presence of Command, etc.</td>
</tr>
<tr>
<td>Timing: B</td>
<td>BTLU - B757</td>
<td>Crate Controller</td>
<td>7</td>
<td>Each line indicates presence of data, etc., from one crate controller.</td>
</tr>
<tr>
<td>Demand Handling</td>
<td>Branch Demand</td>
<td></td>
<td>1</td>
<td>Indicates presence of demand</td>
</tr>
<tr>
<td></td>
<td>BD</td>
<td>Crate Controller</td>
<td></td>
<td>Requests &quot;Graded-L&quot; Operation</td>
</tr>
<tr>
<td>Common Control</td>
<td>Graded L Request</td>
<td></td>
<td>1</td>
<td>As on Datavay Z line</td>
</tr>
<tr>
<td></td>
<td>BG</td>
<td>Branch Driver</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>Initialize</td>
<td></td>
<td>7</td>
<td>For future requirements</td>
</tr>
<tr>
<td></td>
<td>BVL - BV7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5
Signal Lines at Branch Highway Ports
organization of some of the common features, such as addressing and
data lines, and in the handling of demands. There are also two
special timing signals BTA and BTB which play a vital part in the
highway protocol. Table 5 gives the function of the various lines at
each of the branch highway ports, and it is useful to discuss each
of these in turn briefly. In doing so it is vital to remember that
the principal role of the branch highway is after all to transmit
and receive data or commands from the various CAMAC modules, and not
particularly to the crate controller, which should be as 'trans-
parent' as possible.

The first requirement is to be able to address a module and to
define a dataway function. This is done in a similar way to the NAF
command on the dataway by an appropriate combination of BN, BA and
BF signals, but now with the addition of a BCR<sub>i</sub> signal as well to
address the crate containing the required module. These BCR signals
are unique to the branch highway, and there is an individual BCR line
to each crate (i = 1,2,...,7). It is debatable whether there is any
advantage to this scheme, which permits the concurrent addressing of
more than one crate, or whether it would have been better to have a
fewer number of binary-coded crate address lines which would not only
conserve lines but also permit a greater number of crates to be
addressed. In the case of the station number N, there is clearly
only one realistic alternative, namely to use five lines in a binary-
coded address BN, as opposed to the dataway where there are 23 indi-
vidual station address lines. On the other hand, the code provides
for 32 alternatives, and the extra nine codes are allocated as shown
in Table 6.

A very important use of these additional codes arises from the
need to address more than one crate station concurrently. In the
dataway, since there are individual (N)-lines to each station, this
is a possibility which would, however, be lost in a multi-crate sys-
tem because the N is derived from a binary code. Nevertheless, by
using the branch highway codes BN(26) or BN(24) it is possible either
to address all stations concurrently (except the crate controller
station 24) or to address any preselected stations; the preselec-
tion would normally be performed by appropriately loading a 24-bit register in the Crate Controller by means of one of the other extra codes like BN(28).

<table>
<thead>
<tr>
<th>N Code</th>
<th>Use</th>
<th>B, S1, and S2</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>N(0)</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N(1) - (23)</td>
<td>Address the corresponding normal station</td>
<td>Yes</td>
<td>Normal stations occupied by the controller need not be addressed</td>
</tr>
<tr>
<td>N(24)</td>
<td>Address preselected normal stations</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>N(26)</td>
<td>Address all normal stations</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>N(28)</td>
<td>Address crate controller only</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>N(30)</td>
<td>Address crate controller only</td>
<td>No</td>
<td>No Dataway operation</td>
</tr>
<tr>
<td>N(25, 27, 29, 31)</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6
Station Number Codes used in Crate Controllers

In the dataway there are 24-bit read or write signals, carried separately on 48 individual lines. In the branch highway, there are 24 lines used both for the transmission and reception of data. The form of the bi-directional transmission was already shown in Fig. 7. The data lines on the branch highway, designated BRW, are also used for the transmission of L-signals throughout the branch. Since 24 lines are available and there can be a variety of ways to request L-signals, the 24-bit L-pattern can be a composite or even 'graded' version of individual L-signals from modules in different crates.

The status signal (BQ) has the same significance as for the dataway and is just the logical-OR of all the (Q) status signals at
each crate. Similarly the status signal (BX) is analogous to the dataway (X).

As mentioned already there are special timing signals on the branch highway BTA and BTB which do not have a counterpart on the dataway. BTA is used by the branch driver to initiate in any addressed crate a command or a graded-L request; it is generally transmitted after the command, when it has a synchronization function. BTB is a response from any addressed crate to denote that a command has been accepted and that there are either data or L-signals available. Thus BTA and BTB are not only used as timing signals which take into account the asynchronous nature of transmission along branch highways of different lengths, but also as logical signals which indicate that commands have been correctly executed or, in some circumstances, that a given crate is actually physically present or switched "on-line". It must be remembered that it is permissible to address more than one crate concurrently, in the so-called multi-crane addressing mode, and for this reason there are seven individual BTB lines, one from each crate, just as there are seven individual crate address lines BCR. There is a single unaddressed line to all crates BZ, which performs the same function as the (Z) signals on the crate dataway, namely to set the system to some defined initial state. The remaining lines of the branch highway are either reserved for future requirements or with demand-handling. Demand handling is achieved by means of the two lines BD and BG.

4.3.2 Demand Handling

The whole question of demand handling in a system with so many sources of L-signals is a complex one. In principle there can be up to \(7 \times 23 = 161\) sources of L-requests, and each one of these may be a compound signal derived from several sources within a module. In addition, there is provision within the CAMAC specification for logically processing the signals on the 23 L-lines in any given crate before they are further transmitted. This manipulation of L-signals is termed 'grading' in the CAMAC specifications, where it is described rather terrifyingly as "... sorting, processing,
grouping or re-arranging ..." which, I suppose means doing just about anything to the L-signals including leaving them alone!.

Nevertheless, no matter what form the grading actually takes, each crate, through its crate controller, is allowed to generate only one demand signal which is communicated, along with the other analogous signals from other crates, by means of an intrinsic OR connection to the single branch demand line BD on the branch highway. The (BD) signal is directed to the branch driver, and it may occur at any time. The difficulties that there are do not come about because of the (BD) signal, which merely signifies that some attention is required at some combination of stations, but arise in trying to identify the correct pattern of L-requests in an economical way. One way in which this could be done is to scan all the stations sequentially using perhaps the F(8) function, but this is a laborious and generally unnecessary task. The search is usually greatly facilitated by using instead the branch graded-L request line (BG).

The (BG) line is a special one, connected to all crate controllers, which requests that a graded-L pattern (the graded-L word) be transmitted to the branch driver by means of the BRW lines. It must be accompanied by address signals BCR to all on-line crates, so clearly there must be no other data or command transfers in progress when (BG) is issued. The graded-L word may be so arranged that only one cycle of demand source identification is necessary, for example when there are not more than 24 separate L-sources within the branch. Alternatively, it could be that two cycles are required; for example, a first cycle to identify in which crate the attention status was generated, and a second cycle identifying which station in the crate, using N(30) with an appropriate sub-address and function to read out an L-register of 24 bits on to the BRW lines. Clearly, there can be also much more sophisticated schemes when the number of L-sources is greater with, perhaps, varying priority. The correct balance between hardware and software identification of sources of attention status can only be decided in each particular system. Nevertheless, all the commands necessary for this identification are present in the CAMAC system.
4.3.3 The crate controller

It is obvious from the specification of the CAMAC dataway that it is necessary to have a crate controller (CC). Of course, the precise form of the crate controller can be chosen but, even in a CAMAC system with a single crate only, a crate controller is essential, and it must occupy at least stations 24 and 25 of the crate; it must be remembered that, only from station 25, are the (N) and (L) lines accessible, and it would be impossible even to address modules without the controller.

In the case of many crates, it is possible to have a radial configuration and then to use a totally non-standard CC. As has been pointed out, this approach has been adopted in some places, where the CC is used both as a control module and as a computer interface. Nevertheless, bearing in mind the need for a standard for multi-crane operation in the CAMAC system, the importance of the branch configuration has been strongly emphasized, and it follows from this that it is then possible to standardize also the crate controller, at least to a significant extent. In the CAMAC branch, the CC plays the essential role not only of making it possible for the branch driver to gain access to individual modules, both by addressing and (L)-handling, but also for channeling the bi-directional data flow on the BRW lines into the separate (R) and (W) lines of the dataway. These and other features have been incorporated into the CAMAC specifications for a standard crate controller designated Crate Controller Type A-1 (CCA-1). It is a complicated device with many features. Nevertheless, it is useful just to summarize the principal properties of CCA-1, and to emphasize that it is not just a patchboard for passing on dataway signals on to the branch highway, but also a logic device which needs to execute its own commands originating from the branch.

Type A-1 controller is generally a double width module to allow access to stations 24 and 25, and to leave room for panel connectors and it has two important manual features; the first is to set the address (1-7) of the crate into which it is plugged, and the second is to switch the controller into either the off or on-line state.
In the off-line state, there can be no response from CCA-1 to any branch signals and, in particular, both BQ = 0 and BX = 0. The other main features of the controller are as follows:-

i) commutation of (R), (W) and graded-(L) signals on to the BRW lines.

ii) provision for "grading" the received (L) pattern before transmission to the branch highway.

iii) storage of a 24-bit pattern from BRW lines to be used for preselection of required stations; the store is a 24-bit register designated SNR, and it can be loaded from BRW by N(30). A(8). F(16).

iv) provision of timing signals to the dataway and BTB to the branch highway.

v) appropriate decoding of all binary codes (for example, for (BN), (BA) and (BF)) and also some signal shaping and delay.

vi) generate (C) and (Z) appropriate to dataway requirements, either by branch highway commands or manually (push-buttons on CC panel).

In addition to the features just listed, there are also ways to set and interrogate the status of the CC, particularly in respect of demand handling and inhibition; thus there are bistable logic elements which can gate both (I) and (BD), and the state of these elements can be selected and interrogated. Also, just as (BX) could be generated from (X) to signal to the branch the status of given modules, so (BX) must also be generated within CC as a response to any commands executed within CC. A summary of commands actually implemented by CC is given in Table 7.

Most of the preceding description is sufficient to make the function and form of the CC quite clear. There are two points, however, that need to be added. The first is that the actual grading of individual L signals will depend upon the particular application of the CAMAC system, and nothing is said in the CAMAC specification how this should be done. It is usual, however, to perform any
required manipulation of the L signals in a separate module which is connected to the CC by means of a standard connector. Finally, the reasons for the generation of dataway (C) and (Z) inside the CC is because both need to be accompanied by the timing signals (S2) and (B) and, in any case, there is no separate (C)-line in the branch highway; in both cases, the initiating command from the dataway must be accompanied by the crate address BCRj.

<table>
<thead>
<tr>
<th>ACTION</th>
<th>COMMAND</th>
<th>RESPONSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate Dataway Z</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>Generate Dataway C</td>
<td>20</td>
<td>9</td>
</tr>
<tr>
<td>Read GL</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>Load SHR</td>
<td>30</td>
<td>8</td>
</tr>
<tr>
<td>Remove Dataway 1</td>
<td>30</td>
<td>9</td>
</tr>
<tr>
<td>Set Dataway 1</td>
<td>30</td>
<td>9</td>
</tr>
<tr>
<td>Test Dataway 1</td>
<td>30</td>
<td>9</td>
</tr>
<tr>
<td>Disable BD Output</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>Enable BD Output</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>Test BD Output Enabled</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>Test Demands Present</td>
<td>30</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 7
Commands Implemented by CAMAC Crate Controller Type A

4.3.4 The Branch Driver

The necessity for a Branch Driver arises from a general mismatch between the CAMAC branch highway and the characteristics of small computers, even though it is possible to couple CAMAC to a computer in a number of different ways. Perhaps this is not surprising, since the CAMAC system has evolved precisely out of the frustration in using small computers for many applications. Never-
theless, the degree of mismatch between the CAMAC branch and a given computer varies from one computer to another, and for this reason the branch driver must also differ with different computers.

There are several respects in which CAMAC branch characteristics differ from those of I/O structures of different computers. First of all the CAMAC branch requires a particular transmission protocol with specified timing and logical control. Moreover, the CAMAC timing cycle will not only differ, in general, from that of most computers but, in any case, the precise CAMAC cycle time depends upon the physical length of the branch highway and that is quite unknown a priori and so cannot be made a fixed feature of the branch driver. Thus, the very minimum task of the branch driver is to handle the essentially asynchronous nature of highway transmissions and to provide the ancillary timing signals; it is clearly necessary also to make sure that all the signal levels and other hardware standards of CAMAC and computer highways are properly converted.

The second aspect where significant differences exist is in the command structure. Here, the I/O command set of the small computer is certainly much more limited than the extensive repertoire of CAMAC commands, so there is a need for something much more than a simple command conversion. Indeed most of the CAMAC commands generated by the Branch Driver will generally be transmitted from the computer to the driver in a form that is considered by the computer system to be data. Similarly, in certain cases where (BX) and (BQ) are not used by the Branch Driver, the appropriate status bits will be transmitted in a data transfer to the computer, there to be tested and the appropriate action initiated.

At this stage the question of word length arises since in the CAMAC system there are 24-bit data paths and, strictly, the Branch command requires at least 21 bits, namely:

(C) - Crate Address 7
(N) - Station Number 5
(A) - Sub-Address 4
(F) - Function 5
On the other hand, very few small computers nowadays have 24-bit words, the overwhelming majority have 16 bits, while a good number of the smaller process-control computers have only 12. Now there is nothing that can be done about data transfers when the computer word has fewer bits than that of the CAMAC branch, and the only solution is just to make two transfers for the high and low order bits respectively. Nevertheless considerable economy can be achieved in command transfers, by incorporating suitable storage registers in the branch driver; the actual solution in each case will depend upon the precise number of bits in each computer word.

It should be remembered that the basic design principle for a branch driver should be not to impose any further restriction on data transfers than those already set either by the CAMAC dataway or the computer I/O bus or data channel. Thus, it should be possible to execute most CAMAC control operations by performing only a single transfer from the computer, while data operations should not in general require more than two transfers. In order to do this the Branch Driver will almost certainly have connections to enable both programmed I/O operations (usually from an accumulator) and direct access transfers (DMA), although in some small computers (like the PDP-11) there is only one I/O bus for all purposes.

To illustrate the general ideas just outlined, mention can be made of the general approach adopted at DNPL for Branch Drivers, where a different configuration is used for 12, 16 and 24 bit processors; in each case there are STATUS registers for Interrupt and Masking information, together with CAMAC cycle Busy and (Q) status stores. For the 24-bit computer, the solution is simple and there are just two main 24-bit registers, one for data and the other for commands. In the case of 12-bit computers, there are two sets of 12-bit registers for data and commands thus:

\[
\begin{array}{cccccccc}
24 & 13, \\
\text{DATAHI} & \\
C_7 & C_6 & C_5 & C_4 & C_3 & C_2 & C_1 & A_8 & A_4 \\
\text{CONHI} & \\
12 & 1, \\
\text{DATALO} & \\
A_1 & N_1 & N_8 & N_4 & N_2 & N_1 & F_1 & F_8 & F_4 & F_2 & F_1 \\
\end{array}
\]
CAMAC commands are executed by loading both the registers CONLO and CONHI. However, most CAMAC operations do not use a sub-address greater than 112 and successive commands are overwhelmingly without any change of crate address. Thus, in the 12-bit configuration, the majority of control operations can be done by changing the CONLO register contents only.

For 16-bit computers, a different branch driver organization is adopted, with two data registers and a single control register. Now however, there are four crate address registers, each with seven bits.

The configuration is as follows:

$$X_2X_1N_{16}N_8N_4N_2N_1A_8A_4A_2A_1F_{16}F_8F_4F_2F_1$$

CONTROL

$$C_7C_6C_5C_4C_3C_2C_1$$

$$\begin{align*}
  &C_7 \leftarrow \rightarrow C_1, \\
  &C_7 \leftarrow \rightarrow C_1, \\
  &C_7 \leftarrow \rightarrow C_1, \\
  &C_7 \leftarrow \rightarrow C_1, \\
\end{align*}$$

X(0)

X(1)

X(2)

X(3)

CRATE ADDRESS REGISTERS

As in the previous case, with 12 bits, use is made of the fact that most successive commands do not require a change of crate address; thus all that is required is to preload one of the crate address registers appropriately. Actually, because there are two bits in the control register available for crate addressing, there is the possibility of four crate-address registers specified by $$X_1$$ and $$X_2$$.

In addition to the transfers just outlined, it is generally necessary also to specify the starting location in core where data are to come from or go to as well as the number of words to be transferred in the case of DMA operation. This information is usually
contained in registers of the small computer I/O structure, and need not be a part of the Branch Driver. On the other hand, if the block transfer algorithm of Fig. 5 is to be implemented by hardware, using the (BQ) signal, then it is a good alternative to incorporate the appropriate hardware for (Q)-test and (CNA) incrementation within the Driver itself. Clearly also, since the Branch Highway is bidirectional, hardware is necessary within the Branch Driver to recognize the sense of data flow required from the command (F)-bits and to set the (BRW) gates appropriately.

The most complicated question of Branch Driver implementation arises in meeting the requirements of demand handling, partly because the interrupt-handling properties of computers are generally different from that required for CAMAC and partly because interrupts can arise not only from the (GL) and (BD) lines, but also from other sources such as (BQ) and (BX). As already mentioned, demands can arise in CAMAC either from the (BD)-line, or from the graded-L pattern (24-bits) from all crates in response to a (BG) command, or from an L-Register in a CC in response to N(30) and an appropriate function code, or, even more precisely, from the (BQ) response to an F(8) function (Test L) to each sub-address capable of generating (L). Because of these very comprehensive possibilities, the only universal thing to do in a Branch Driver, in order not to be too restrictive yet to be able to rapidly identify an interrupt source, is to provide within the Driver a 24-bit GL register, single-bit (BQ), (BX) and (BD) stores and an interrupt mask register, and then to make all these accessible to the priority-interrupt system of the small computer. The precise sequence for demand handling can then be decided in each application, and it must be remembered that this sequence must include resetting of the appropriate (L)-flags as soon as the relevant demands have been serviced.

4.3.5 Multi-branch Systems

The problem of implementing CAMAC systems with more than seven crates, and hence with more than one branch, has not been considered in the CAMAC specifications. Because of the addressing structure of the Branch Highway and also because of practical difficulties of
driving more than seven signal-sinks along the highway, it is not an easy thing merely to extend the branch to more crates. And, although seven crates full of CAMAC modules represents a very large system indeed, there are already applications which call for an even greater number of crates. Thus there is a need for CAMAC systems with more than one branch.

It is still too early to make any firm recommendations about multi-branch systems, but one general approach which does not violate the CAMAC specification is simply to have a separate Branch Driver for each Branch, and then to interface each driver to a separate data channel of the control computer. Another alternative is to use a single branch driver with internal multiplexing facilities, but then great care has to be taken to correctly identify demand sources.

4.3.6 Typical transfer times

It may be thought, with all the complication of the CAMAC dialogue, that the time taken to transfer data words to or from a small computer will be rather long. However, by incorporating both DMA and program-controlled transfer facilities in Branch Drivers, and by the use of intelligent features, such as incorporating a separate command register in each Crate Controller to reduce the overheads of the Branch Highway protocol, it is possible to achieve CAMAC transfer times which are very little worse than those of the CAMAC cycle itself.

A typical example of what can be achieved in the CAMAC system using most types of processor with about 0.75 μs cycle time is the following:

1.2 μs - for 16-bit words successively read into memory via DMA (not including overheads);

2 μs - for each 16-bit word read from say 10 adjacent modules, each with four sub-address registers, and storing all data into memory; this time includes all the overheads;

12 μs - to read a full CAMAC word (24-bit) from one module and write it into another;
20 us - to get a 24-bit graded-L pattern from any crate into computer registers as a response to a Branch interrupt (BD); the time includes identification of source crate.

4.4 CAMAC Modules

Within the CAMAC system, in addition to the Dataway, Crate Controller and Branch specifications, there are also conventions for the signals (both digital and analogue) that can be transferred through the front panel of modules, and the physical form of modules are also very closely defined. Nevertheless, it is impossible to catalogue all the different types of modules that may be employed within CAMAC systems, although it is worthwhile mentioning one or two general points to illustrate the flexibility of the CAMAC system.

The first point to make is that, although CAMAC is principally useful for connecting equipment to a computer where there is some need to transfer digital patterns in either direction, it does not follow that it is necessary to build all the equipment in the CAMAC format. The first reason for this is that the cost of a CAMAC crate space even of single width is quite high, because of the necessity of dataway connection and also power supply. Therefore, it is best to minimize the width of any given module to the minimum necessitated by the requirement of power and dataway connection (or number of required sub-addresses). The second reason is because the total number of stations in each branch is strictly limited, and it is unwise to occupy a station unless there is either spare space or if there must be a close physical association of equipment.

To illustrate the preceding point two simple examples can be given. In the first, we can consider multi-channel analogue-to-digital converters, which are well suited to CAMAC implementation, both because the digital pattern inevitably requires computer processing and because such ADC units are generally only part of some modular system whose precise configuration depends upon the application. Here, it is clear that there is no necessity in the case of very many channels (e.g. 4096), to put all the hardware in one CAMAC module which may occupy as many as say eight stations; a
single-width CAMAC module can be used which is interfaced to the rest of the hardware built in any required format (e.g. NIMS). As a second example one can point to manual controls, such as switches, patchboards or keyboards, or to visual indicators, such as lights, read-out displays or graphics terminals. In each of these cases, although it makes very good sense to interface all the various units through CAMAC, the most economical solution is to build them actually in the most convenient way ergonomically, with care to minimize cost, and then to connect each unit to appropriate single-width CAMAC modules.

The second point that is worth making is really a corollary of what has just been said, and it concerns the digital control of complex instrumentation systems. In nuclear physics, both before and after the introduction of CAMAC, there have existed systems of complex instrumentation which it has been necessary to control in some sense or other. A good example of such a system is the triggering logic and data acquisition hardware for counter experiments in elementary-particle physics. In general, such systems have not previously had all the facilities appropriate for digital control, and solutions to the problem had been made by developing special-purpose hardware. With the advent of CAMAC, a new and elegant possibility of digital control was introduced. However, it is important to remember that in order to use CAMAC for digital control of these complex systems, it is not necessary to rebuild the whole system in the CAMAC format, but only those parts which are related to the flow of digital information to the control processor. The rest of the system can remain unchanged, except for appropriate interfaces between control gates and registers and the relevant control-data registers somewhere in CAMAC. For this, all that is necessary is a set of CAMAC modules which enable the input or output of sets of either high-speed or low-speed logic signals. Such modules are termed pattern units. For example, at Daresbury, CAMAC pattern units have been developed to input or output either 16 programmable fast logic levels to the NIM specifications (single-width) or, alternatively, 256 signals of TTL levels (double-width). These units have proved to be of immense value - which is perhaps
not surprising since they allow the dynamic logic re-configuration of quite complex systems at intervals of 20 µs or even less.

4.5 Software for CAMAC

In a certain sense, the CAMAC system offers to users hardware facilities which are analogous to the programming facilities of a high-level language. That is to say, like a sub-routine or procedure, a CAMAC module is interchangeable, it has a well-defined interface to other parts of the system, and it enables users to exploit the facilities of the modules without understanding any details of the modules themselves. However, there the analogy ends, because it is still necessary for most CAMAC users to know both the Assembly language of the control computer and the conventions of the CAMAC command structure in order to actually use any given CAMAC module. And, as has already been indicated, the CAMAC dataway and branch highway protocols are not uncomplicated.

It is not surprising, in view of all the powerful hardware properties of CAMAC, that attempts should be made to define and implement a language to enable modules to be used without the necessity of coding every CAMAC command in the equivalent of a computer machine language instruction. Nevertheless, the question is an extremely complex one, because of the very large range of possibilities in configuring CAMAC systems; for example, it must be recalled that one of the possibilities allowed within CAMAC is the autonomous transfer of words from one module to another.

At the present time there is no formally-defined CAMAC language. However, there are several different approaches which have been made in the direction of producing a higher-level means of communicating between CAMAC and control computers.

There are several possibilities for CAMAC conversations:

i) Normal CNAF commands and Graded-L handling where both are treated as data transfers from core locations; the transfers are made by standard I/O features of the assembly language.

ii) Treatment of all CAMAC registers (command, data and graded-L) as extensions of memory addressing structure enabling CAMAC
to be programmed in the same way as any other memory locations (or registers) within the control processor; this approach is a natural one for PDP-11 or similar architecture, where control, status and data registers of peripheral devices are all directly addressed as memory locations.

iii) Incorporation of CAMAC data and command handling features within an interpretive language at a fairly high level; usually by modifying the reader and punch sections of the language processor.

iv) CAMAC communication by special purpose routines in a high-level language, but using the standard syntax features of the language (e.g. sub-routine CALL or Procedures).

v) Implementation of appropriate new MACRO procedures as an addition to the standard assembly-language repertoire.

vi) Definition of a new high-level, universal CAMAC language.

It is impossible to discuss in detail all the possibilities just listed, mainly because there is just not enough experience of the CAMAC system itself yet, far less is there experience of CAMAC software; we must remember that the specifications of the branch highway have only been finalised in mid-1972. Nevertheless there are one or two general points worth making, and it is also worthwhile reviewing the progress that has been made in one important practical area, namely the development of a Macro language for CAMAC.

The first point to make is that the majority of CAMAC systems implemented so far indeed use the first programming technique listed above i), namely to form CAMAC commands from appropriate bit-strings and to transmit them from computer to branch driver by data transfers. It is already clear that this is a tedious procedure which is highly prone to errors in programming.

Consider for example the assembly code that would be necessary just to perform the following simple operation on a data register at sub-address 3 in module station 7 in a crate whose address is specified:
1. Wait for device to be ready.

2. Write a 16-bit data word from a symbolic location NICHE in memory.

Let us assume that the program is to run on a Honeywell DDP516, and that the Branch Driver has a 16-bit control register.

\[ X_2X_1N_{16}N_{15}N_4N_2N_1A_8A_4A_2A_1F_{16}F_8F_4F_2F_1 \]

Here, \( X_2 \) and \( X_1 \) specify one out of four auxiliary crate address registers; in this example we can assume that the requisite crate address is stored in auxiliary register 2. We need to know that the CAMAC procedure for writing a data word is \( F(16) \), that \( F(27) \) will test the sub-address status and generate \( (Q) \) if the device is ready, and that the relevant DDP-516 Assembly codes are as follows:

- **LDA** Load Accumulator
- **OTA** Output from Accumulator and skip next instruction if device was ready.
- **SKS** Skip on condition
- **JMP** Jump
- **DEC** Define Decimal Values
- **OCT** Define Octal Values

The required assembly code is:

```
LOOP  LDA  'FUN27  Load Accumulator with X(2)N(7)A(3)F(27)
       OTA  '240  Move to control register - Execute CAMAC cycle
       JMP  *-1  Loop if busy
       SKS  'TSTQ  Skip if \( (Q) = 1 \)
       JMP  LOOP  Try again
       LDA  'NICHE  Load Data from symbolic address
       OTA  '340  Output to data register
       JMP  *-1  Loop if busy
       LDA  'FUN16  Load Accumulator with X(2)N(7)A(3)F(16)
       OTA  '240  Move to control register - Execute CAMAC cycle
       JMP  *-1  Loop if busy

* FUN27 OCT 107173 \( X(2)N(7)A(3)F(27) \)
FUN16 OCT 107160 \( X(2)N(7)A(3)F(16) \)
```

Of course, the actual coding of the above example could be greatly simplified by writing CAMAC I/O subroutines for use within the assembly language. To illustrate this the previous example
would now become:

```
LOOP    CALL CAMAC
DEC 0    Defines 16-bit transfer
DEC 2    Crate address register
DEC 7    Station
DEC 3    Sub-address
DEC 27   Test status command
CALL CAMAC
DEC 4    Defines Test (Q)
JMP LOOP
LDA NICHE Load data
CALL CAMAC
DEC 0
DEC 2
DEC 7
DEC 3
DEC 16
```

and, in practice, this would be written in the more contracted form:

```
LOOP    CALL CAMAC
DEC 0,2,7,3,27
CALL CAMAC
DEC 4
JMP LOOP
LDA NICHE
CALL CAMAC
DEC 0,2,7,3,16
```

The above example indeed illustrates that many of the areas of possible error in coding can be eliminated and drudgery reduced. Nevertheless, the resulting implementation leads to very slow execution, requiring typically 100 µs for each CAMAC operation, so the range of applications is quite limited.

Although the second alternative mentioned earlier ii), of treating CAMAC registers as extensions of memory locations, offers a more unified programming structure, it is still a low-level approach and it still would be necessary to make sure that the transition to high-level (for example, by addressing the relevant CAMAC registers by names or as elements of arrays) could be made without destroying the "real-time" aspects of CAMAC hardware.

The third alternative, to use an interpretive language, is an attractive approach for certain types of problem, and it has the advantages of easy program development and the fact that many users
are already familiar with certain of these languages. This fact has been used by certain groups, who have developed an overlay for FOCAL, for example\(^4\), which allows the incorporation of CAMAC statements in the language, and it stores and handles CAMAC data like normal variables. FOCAL is a conversational programming language developed for use with DEC machines.

An example of a CAMAC statement in the modified FOCAL, which will write a CAMAC data word into an address CNA, using a function F, is:

\[
*_{\Delta 7,C,N,A,F,B,W,L,W,Q}
\]

where HW and LW are the most and least significant 12-bit components of the data word and FB is a format byte which defines the format (decimal, BCD, OCTAL) in which data and commands are handled. Another example, to read the attention status (L) from a module (M) in a crate C, is simply

\[
*_{\Delta 5,C,N,R}
\]

where R = 1 if (L) = 1.

So it can be seen that the above approach is a very straightforward one, because it avoids the laborious task of developing a completely new programming system, and the syntactical form that must be used for a CAMAC dialogue is close to the CAMAC hardware. The overwhelming disadvantage of this alternative is, however, that it is very slow and there are fundamental problems in handling fast interrupts. Nevertheless, the system is of very real value in execution of test routines and in system development.

The use of standard features of high-level languages, for example, by using subroutine calls, is one which appears to be a very promising one from the user point of view. Nevertheless, although the incorporation of data and command transfers is not an insurmountable task for most high-level languages, the handling of interrupts and the adherence to strict timing requirements presents very severe problems indeed. Attempts have been made, in languages such as "real-time" FORTRAN on certain machines, to overcome the obstacle, but there have been no successful incorporation of CAMAC
features into these languages at the present time. In a similar way, no 'universal' high-level language has been developed for use with CAMAC, although there is much discussion of the problem. It must be remembered also that the implementation of a high-level CAMAC language is not just a linguistic and compiler question but can also be a problem for the operating system, since a CAMAC program is generally mainly concerned with input and output to a computer peripheral. On the other hand there is no reason nowadays why the use of a high-level language on a small computer should be inhibited by the lack of core store and other facilities, since it is now very common practice to use a large computer to actually perform the compilation and produce the requisite object program modules for the small computer.

Although there may be developed eventually a high-level language for CAMAC, probably the best solution for the present time (and for the next few years also!) is the alternative v) in the list given earlier, namely to use a set of specially defined Macro procedures within the Assembly language of the control computer. In this way, by a careful choice of Macros, a CAMAC program can be made up mainly of instructions which an unskilled programmer can easily remember and use, with a minimum of actual assembly code instructions. Such an approach is particularly well matched to a system where the small computer is used as a CAMAC controller, but is linked to a larger computer used for data manipulations and reduction; in such systems, the small computer is employed dominantly for data acquisition, demand handling and control, for which purposes a Macro language is eminently suitable.

To illustrate the power of a Macro language, the source code required to perform the example previously given in connection with programming in Assembly language would be simply:

```
@TSQ    DEVICE    Test Q
LDA     NICHE     Load Data
@ST1    DEVICE    Write Group 1 register
```

where DEVICE is a symbolic name for the address X(2)N(7)A(3). Now however, unlike the previous example of contracted notation using
I/O subroutines for CAMAC transfers, the code produced by the Macro processor would be identical to the hand-coded assembly routine and so would be just as fast. It would be difficult indeed to produce a more succinct notation for CAMAC, even in a high-level language.

The foregoing example was given using some of the facilities of CAMACRO, a language developed at Daresbury\(^5\) for the DDP516, H316, H112 and PDP11 processors. CAMACRO defines the syntax and conventions of language features added to the Assembly language. It is implemented by modifying the standard Assembler both to translate the CAMAC Macro statements and also to enable strings of CAMAC arguments to be defined. Work is also very advanced in the development of a CAMACRO pre-processor, which will take source code and replace all CAMACRO calls in such a way that the resulting, pre-processed code can be translated by the standard Assembler, either on the small machine or on large systems such as an IBM 360. In either case, the modifications are such that anyone can continue to use the Assembler in the normal way without using CAMACRO statements.

The principal feature of CAMACRO is that it allows programmers to use Macro names which are close to CAMAC functions, for example \$F08 will produce the code to execute a CAMAC cycle with function F(8). In certain cases, it is also allowed to use alternative mnemonics; for example \$TLM can be used in place of \$F08 to denote a Test 'Look at Me' command.

Each Macro can have several arguments or parameters, for example, to define CNAF codes and (Q). However, unlike most conventional Macro commands, in which parameters are "simply substituted", CAMACRO requires a packing definition as well, so that the parameters are located at the correct bit positions in CAMAC commands. To illustrate this point let us consider some Macro whose name is \$MACR which requires three arguments in a CALL, for example

\[ \$MACR \, \$1,\$2,\$3 \]

Then, if this Macro produces four lines of assembly code with a total of two parameters \#1 and \#2, a typical definition of \$MACR would be as follows:
$\text{MACR}$

$\#1 = [1,2,31] [3,7,32] [8,11,33] [12,16,9]$ 
$\#2 = [1,2,31] [3,7,10] [8,11,33] [12,16,0]$ 

$LDA = \#1$

$STA \quad LOC1$

$LDA = \#2$

$STA \quad LOC2$

where each of the 16 bit strings $\#1$ above involves brackets of the form $[\alpha,\beta,n]$, and $\alpha$ and $\beta$ are the lower and upper bit bounds and $n$ is either an argument or a literal quantity. In addition to the above facilities, each new Macro can use either the code or the string definitions (or both) of previously defined Macros.

4.6 CAMAC systems

It would be impossible to survey all the applications of CAMAC systems, because of the universal properties of such a powerful modular system. Nevertheless, there are one or two broad areas where CAMAC has had a profound impact, and it is useful just to mention these briefly.

There is no doubt that the problem of reading out blocks of registers has been solved very effectively by using CAMAC. Prior to CAMAC, this problem was tackled in many different ways, each with its own restrictions and special features. To illustrate the situation, we can consider the most common problem encountered, namely that of reading out blocks of scalers. In previous implementations, there was always the question of scaling format and of visual display. Users wished to have the option of displaying the contents of particular scalers, and this generally leads to BCD scaling formats. As a result, the stored numbers were unsuitable for eventual computer processing, and time-consuming conversion became necessary. In any case, the transfer of blocks of scaler data to computers presented other problems, since there were no standards, and data was generally written on paper or magnetic tape, again with compatibility problems. The CAMAC system has solved the problem of scaler readout very effectively, since the scalers can now all be binary, and hence cheaper, more compact and with data suitable for computer processing.
The question of display in a CAMAC system is trivial since any required format conversion can be done either by the control computer or by a suitable display module, and the scalers to be displayed are chosen either by a manual selection module or by the standard computer input keyboard. Other advantages of the CAMAC system in this application are that system checking is easy to introduce and any new test requirements rapidly incorporated, and that the system can be easily expanded either to include more scalers or indeed blocks of other registers (such as pattern or parameter units). It is easy to see that CAMAC has removed many of the restrictions previously associated with specially implemented block readout systems, and has introduced a tremendous degree of flexibility while at the same time making it possible to carry over systems from one computer to another with a minimum of effort.

In a similar way as for scaler systems, CAMAC is most applicable to the readout, manipulation and display of data from other devices with digital output registers, such as analogue-to-digital converters. It is precisely this possibility that is making CAMAC so useful for such systems as multi-channel pulse height spectral analysis. Similarly, more and more CAMAC systems are being used for the readout of digitizing spark and proportional chambers.

Another area where CAMAC has been of immense value is in a relatively new application, namely in calibrating system parameters, particularly in the triggering logic for counter experiments. An example of such an application is in a system called ACE\textsuperscript{6}, as mentioned an acronym for automatic calibration of experiments. In the ACE system, all the EHT and signal leads to scintillation counters, for example, are multiplexed so that they either connect to their intended locations in the triggering logic and auxiliary equipment (such as power supplies), or a selected channel can be modified and controlled by the adjustment of several parameters of interest; examples of such parameters are signal delay, EHT, attenuation and so forth. Then, by the use of real particles or, more usually, by employing light-emitting diodes (GaP junction diodes) built into the counters themselves, the correct operating
values of parameters can be established.

As an example of a single-channel calibration, Fig. 8 shows the
EHT characteristic of a photomultiplier in a scintillation counter.
The required characteristic is obtained simply by supplying a known
number of calibrated light pulses for each selected value of the EHT.
The resulting curve can be visually displayed or just stored in com-
puter memory, and the required information can be easily derived
from it. In this case, the two important facts to be determined are
first that a plateau exists and second that the plateau is at 100
percent counting efficiency; the correct operating value of EHT can
be selected along the plateau.

Another example of the usefulness of the ACE system is in de-
termining delay curves, which is done by choosing a reference chan-
nel and then varying the relative signal delay between it and any
other selected channel. The resulting characteristic curve, shown
in Fig. 9, shows the required, symmetric form with 100 percent
counting efficiency; the correct value of operating delay can easily
be defined from this delay curve, either by a simple computer
algorithm or by eye.

For both the examples just given, the ACE system was implement-
ed mainly in the CAMAC format. This allowed the system to be
assembled very easily from standard modules, and the associated
programming task was very straightforward. An important feature of
the system was the visual display which is also interfaced using
CAMAC modules. There is no doubt that the use of the ACE philosophy
can dramatically reduce much of the routine necessary for setting
up complex counter logic systems. It can also be used to check that
such systems, once set up, remain correctly operating.

Extensions to the ACE system are obvious, and such systems are
now in widespread use throughout counter physics. A very recent
example of the technique is in one of the experiments to be run with-
in the Omega project at CERN, where the principal trigger criterion
in a missing-mass experiment is that there should be a slow recoil
proton. The proton can cover a wide angular range, so the proton-
trigger counter is a set of large scintillation counters about
Fig. 8  EHT Characteristic

Fig. 9  Delay Curve for Two Channels
10 cm wide and more than 1 m long. It is necessary to know the approximate position of the proton along the counter in order to determine the recoil angle, and this is found by the fairly well-known technique of having a photomultiplier at either end of the scintillator and then recording the relative arrival times of both signals. Unfortunately, because of the very restricted space available within the Omega magnetic-field volume, there is no room to attach the photomultipliers directly at both ends and it is necessary to have them both at one end with a folded light guide behind the scintillator; the resulting configuration is shown in Fig. 10.

![Diagram](image)

Fig. 10 Section through recoil proton counter segment

A difficulty now arises because spurious particles (typically high-energy pions) going right through the scintillator can also generate Cherenkov radiation in the light guide, thereby giving unwanted signals in each photomultiplier. The problem is to eliminate the spurious pulses, which is not entirely simple. The ACE technique
was used to study the effect of amplitude and time discrimination, using a test beam of ionizing particles and moving the trigger counter to cover all the working area. A typical graphical display is given in Fig. 11a, which clearly shows a beam profile in the scintillator, together with the fainter, spurious profile due to Cherenkov radiation in the light guide. Fig. 11b shows a resulting distribution of trigger pulses with a different discrimination threshold. In this example, the essential components of the calibrating system are a small computer, a graphical display, and the ability to control all the variable parameters rapidly and simply. All these properties are readily available in the ACE system by using CAMAC.

The last example of systems applications of CAMAC that is worth mentioning is in computer peripheral interfacing. The traditional method of connecting computer peripherals to process control computers is by special-purpose interfaces attached to the I/O channels. In this case, the solution is generally costly and quite peculiar to each particular computer. Moreover there is usually equally special-purpose software to control each separate peripheral device. Thus the traditional approach to the problem of attaching peripheral equipment is very inflexible since the special features required generally have to be foreseen at the time of initially ordering the computer, and the cost is such as to inhibit the attachment of all but the bare essentials (particularly in the case of very small processors).

CAMAC offers a very attractive alternative to the manufacturers interfaces for attaching peripheral equipment, and this is a possibility which is being adopted more and more by users who have some good reason to use CAMAC anyway. In the CAMAC solution to the problem, there is only one interface to the computer, namely the CAMAC Branch Driver. All the peripheral units, whether they are special experimental instrumentation or standard I/O equipment, are connected to the computer through CAMAC modules. Moreover, any special peripheral features can be added by incorporating further CAMAC modules. Thus paper-tape readers and punches, alphanumeric displays, key-
Fig. 11(a)  Distribution of pulses in proton trigger counter
Fig. 11(b)  As in 11(a) but with amplitude discrimination
boards, hard-copy devices, magnetic tape units, storage-tube graphical displays can all be interfaced using CAMAC modules which are readily available. Furthermore, the addition of optional features such as hardware generated characters or vectors for graphic display can be simply effected by just incorporating further modules in the system. There is no doubt of the power and flexibility of the CAMAC philosophy for interfacing peripherals. It must also be remembered that any developments in this way that are applicable to one computer can be easily transferred to another provided that it has a CAMAC system. What is perhaps surprising is also that the cost of interfacing using CAMAC is typically no greater than using manufacturers own equipment and, frequently, it is less expensive.

One of the advantages of using CAMAC connected peripheral devices, that is perhaps not so obvious, arises if a language like CAMACRO is used. Then, not only are the peripheral devices themselves interchangeable from one computer to the other but so, to a significant degree, are the software packages to drive them.

To illustrate this point, we can consider the situation at Daresbury where not only are there CAMAC interfaces for all small computers in the laboratory, but there is also the convention that the lower numbered stations in the first crate of each CAMAC system are reserved for standard peripherals used with utility software. In this way, string definitions for the addresses of the relevant devices can be included in the appropriate Macrc definitions. For example, $RDR$ is the symbolic name which automatically addresses a paper-tape reader in station 5 of crate 0. Thus a program segment to read a block of data from the paper-tape reader into an array called AREA on a Honeywell DDP 516 would appear as follows:

```
LOOP  $F27  $RDR,0  TEST FOR READER READY
JMP    LOOP  LOOP UNTIL READY
$FO2  $RDR,0  READ DATA
STA    AREA,1  STORE DATA INTO ARRAY
IRS    0  INCREMENT INDEX REGISTER
JMP    LOOP  REPEAT IF INDEX NOT ZERO
....  CONTINUE IF INDEX = 0
```

Clearly it is a very much simpler task to write and use the above code than to write the required program in assembler language!
5. **DATA LINKS**

In considering the system aspects of small computers in elementary particle physics it is impossible nowadays to exclude data links. This is because the systems consist of several components, and data links are the indispensable means of interconnecting many of these system components together; we have already discussed some of these components earlier, such as CAMAC and indeed the small computers themselves.

There are many ways of describing the role of data links. For example, they can be seen as a way of extending the power of a small computer by coupling it to a larger processor. Alternatively, they may be considered as the means of using a certain fraction of the power of a large processor by a remote user. In either case, it is difficult to talk about data links without involving small computers. And, as we shall see, this is not only because at least one of the terminals to any given data link is generally a small computer used for some application, but also because small computers are themselves used for purely system reasons as well.

Of course, the term data link appears somewhat restricted, but it will be used here in the broadest sense to include the communication not only of data but also of control information, display files and even programs; as we saw in the discussion of CAMAC, it is difficult to make any distinction between real data and command transfers.

In what follows, I shall not try to give a comprehensive survey of the use of data links everywhere in elementary particle physics, but rather I will describe the applications and principal features of data links in one particular laboratory, namely Daresbury. In doing so, I will emphasize the particular philosophy adopted there, that of "distributed computing", where the computing facilities required for any particular application are distributed among the available processing resources by means of a data-link network; in this way there is a good chance that each separate computational task
will be handled by the system resource best matched to meet the need.

5.1 The need for data links

Before discussing any concrete implementation, it is as well to review the applications that give rise to a requirement for data links. Such applications are not difficult to find, but there are more and less conventional ones. Perhaps the most usual need for links arises in making use of normal features of any modern operating system, such as remote job entry, file manipulation and conversation-al debugging either of programs (the location of "spelling" mistakes and simple logical errors) or of arithmetic routines (the so-called "desk calculator" application). These applications rarely require any special hardware or software.

Other applications, which are not so conventional, include data acquisition, management information retrieval, low-speed process control and local interactive graphics. The first example appears to be a very obvious one indeed when the data come from a counter experiment and are buffered by a small computer. The data are certainly due to be processed by a large computer sooner or later and, even if archive storage on magnetic tape is needed, the sooner that the data are transferred to a large system the better. The best way to do this is to couple the small experimental computer to the large system by means of a data link. However this is still not a widely-adopted technique, and for this reason I have described it as "not so conventional". The majority of counter groups use local magnetic tape to buffer their data, but I am confident that this method will gradually disappear. By contrast, at Daresbury, every counter experiment has its own data link and, for several years already, no local magnetic tape buffering has been used; indeed some groups no longer have any local magnetic tape unit at all, even as an "insurance" against link failure, since the link is much more reliable (and cheaper:) than the tape drives. Of course there is a need not only for experimental data acquisition, but also for "engineering" data, for example data from magnetic field measurements, and this becomes of particular importance in control applications.

Mention was made both of information retrieval and local inter-
active graphics. However, these differ mainly in that the first application generally requires the display of alphanumeric data only, since both nowadays need the use of an interactive graphics terminal with at least a display and a keyboard. Now both these applications are well known to most manufacturers, who provide both terminal equipment and associated software. A difficulty arises, however, in that the manufacturer’s solution is generally very costly, and that although a large laboratory can afford one or two fully interactive graphics terminals (of the IBM 2250 type, for example), it is difficult to have more; there are similar constraints in the area of alphanumeric terminals. In these applications, a solution can be found by using data links to attach special terminals, in this way removing most of the practical restrictions.

In the case of low-speed process control, data links can also be of great value not only, as mentioned, for the acquisition of the relevant data, but also for the feedback of control information either directly or through some sort of display. This is a growing application which has been stimulated not only by the availability of powerful processing facilities by means of data links, but also by the existence of CAMAC as a modular equipment interface. Such applications were previously inhibited because of the necessity, for each new control application, to provide dedicated local processing facilities and special-purpose hardware and software.

We come now to the final area of applications of data links, which I would describe as unconventional. Here I would include two broad cases, the first being remote (long-distance) interactive data processing and the second is high-speed control. In the control application, I have in mind, as an example, the possibility that data from a counter experiment might be processed sufficiently rapidly (perhaps on a sampling basis) that control information could be fed back to the experiment on a time scale sufficiently short to be able to usefully modify either an experimental or instrumental parameter. Clearly such an application implies the existence of very fast data links. The other application I mention can be simply summarized as the requirement to use all the facilities of a large central computing system remotely, at an arbitrary distance away
(perhaps several hundred kilometers), as if the user were near the central facilities. Here there are special data link requirements involving completely new developments of buffering, coding and error recovery. As we shall see, the small computer has an important part to play in these developments.

5.2 **System features of data networks**

Bearing in mind the range of applications just outlined, a list of characteristics required of data links can be made. The first requirement is that there should be standardization and modularity, to avoid the need to implement different data links for each application, and also to be able to interface to different terminal devices, often several concurrently, without having to develop much new hardware or software. Another requirement, coupled to the first, is that the network should be reliable. Not only does this mean reliable hardware, but it also implies sophisticated techniques for error recovery and "fail safe" or "fail soft".

A most important requirement in a data network is that it should be possible to have many concurrent users of a central processing facility, for clearly it would defeat the purpose of the network if there could be only a single user at a time. In a similar way, it would be most restrictive to allow only one terminal device on each link, and there must be means to operate several devices concurrently without mutual interference. Nevertheless, it is essential, in meeting all these requirements, to satisfy the most important need, namely to have system transparency. In other words, a user must be able to communicate with terminal equipment by programs in the central computing system, without any knowledge of the details of the data network.

If all the above characteristics are taken into account, a set of system features can be formulated for the data network. At Daresbury, the principal system features for the "domestic" network (namely that within the laboratory boundaries) are as follows:

i) High speed links only (channel rates).

ii) Front-end processor for buffer-multiplexing.
iii) Use of all links through standard high-level language modules within the normal operating system (MVT).

iv) Multiplexing at the user terminal.

v) Professional hardware and software implementation.

The first of these features comes about for two reasons, principally because of standardization and economy, but also because many communications transfers can be handled much more effectively if they occur in blocks at core or channel rates (say 1 Mbyte/s). In this way, the overheads that inevitably occur as a result of demand handling and message switching in a multi-source multi-sink environment can be minimized.

The second features, namely the use of a front-end processor, is a natural decision, bearing in mind that nearly all the links at Daresbury terminate at the central computer. This radial configuration nevertheless emphasizes the deficiency of large systems for handling high-speed data communications, and it turns out that it is much better to divide the link-handling task into two parts, processing and demand-handling. The first is best satisfied by the central machine, while the second, which includes buffering, multiplexing all external lines, communications error handling and data blocking, is best handled by a dedicated processor, the so-called "front-end" processor; it is not surprising that such front-end or communications processors are just small computers with normal features of parallel data channels and priority interrupt system.

The use of the data network by means of high-level languages is clearly one of the most universal methods of obtaining system transparency for the user. This feature cannot be stressed too highly, for it enables a user to communicate with terminal equipment without knowing any of the details (or indeed of the existence) of the links themselves, the front-end processor or the activity of other users.

The need to multiplex at user terminals is a natural one since there is a broad range of applications. However, this need is generally satisfied by defining a single modular interface only and, in this particular example, CAMAC has been chosen. Nevertheless, in
the event of very universal applications, such as graphics, there could be a case for allowing the direct attachment of appropriate terminal equipment to a data link. In the case of CAMAC, by far the best way to connect to a link is through a small computer, since otherwise both the complexity and cost of the appropriate branch driver would be comparable to that of the smaller processors.

The last system feature listed, that of professional hardware and software implementation, hardly needs to be mentioned, and it is the only way that real system reliability and flexibility can be attained. Nevertheless it implies that there should not be, as is so often the case, separate ad hoc solutions for each special application.

5.3 Practical link implementation and user terminals

The practical implementation at Daresbury of the system features just outlined has involved a number of components. First of all, there is only one single standard wire link for all branches of the network within the laboratory. This link is physically a multi-core twisted-pair cable, which can be of any length up to several kilometres, and the main performance characteristics are transfers of 16-bit parallel words at a rate of up to 10 Mbits/s with full duplex operation. The link terminal units are standard, one of which is shown in Fig. 12 and almost all the interfaces are the same CAMAC module.

All the links are connected to a front-end processor, in this case an IBM 1802, which is connected in turn to the central computer, an IBM 360/65, by means of a channel to channel link. As previously mentioned the front-end processor is concerned only with demand-handling, data-blocking, error checking and multiplexing; there are no applications programs. In order to do this, a multi-programming operating system has been written of considerable complexity, although this is a relatively fixed set of programs with defined buffer areas in core. In addition the 1802 processor performs a despatching function and acts as a communications terminal for the 360. Nevertheless the 360-1800 relationship is that of master-master, since either one interrupts the other to initiate the appropriate
Fig. 12  Data Link Interface
service, in this way avoiding the necessity of either processor polling the other. At the 360 end of the link there have had to be modifications to the operating system (MVT) to handle the 1800 link traffic.

Although the 360 system is very reliable, it is not inconceivable that a 360-1800 link failure could occur. It is worth mentioning therefore that the 1800 can perform the useful role of a buffer device to enable at least the "spooling" of input data onto magnetic tape. This is an elementary form of "fail-soft" system which is further enhanced by an alternative link entry into the 360 by means of a 2701 parallel data adapter. Unfortunately the 2701 is a non-programmable device and is consequently very restricted in its application.

The final component that should be mentioned is CAMAC. As already indicated, CAMAC is an extremely useful interface not only for experimental equipment but also for peripheral devices such as graphics terminals and keyboards. In using the wire links just mentioned, it is clear that CAMAC can be readily attached using a small computer not only as the link terminal but also as the CAMAC system controller. This is an extremely powerful combination, since users are now enabled not only to transfer blocks of data directly from CAMAC modules (or sets of modules) into a FORTRAN array, but they may also transfer data back to CAMAC. In the reverse transfer from central computer to a terminal, it must be remembered that, provided there is suitable translation software, a data file may represent control data, or it could be a display file, or it could even be a CAMAC command segment. To illustrate this point it can be mentioned that there exists for the 360 an Assembler which produces object code for a Honeywell DDP-516, and there is also a CAMACRO facility; they can be used together in the 360 to send CAMAC programs via a data link to be executed directly on a terminal DDP-516.

The range of terminals that can be connected to the wire links is quite large although, principally, the links terminate in a small computer such as PDP-8, PDP-11, H316, DDP-516, ARGUS 400, H112. In general CAMAC is interfaced using any of these small computers.
Graphics devices can be attached in any one of several ways, either directly (sharing a link with other devices by means of multiplexing hardware), or by means of a manufacturer's interface to a small computer, or through CAMAC. The last possibility is a very useful one since modules exist, for example, to drive storage-tube terminals in point-plot mode and then to generate characters and vectors by hardware using additional modules. In the case of alphanumeric terminals with graphical display, the CAMAC system has been used simply as a modular multiplexing system, and all such terminals at Daresbury are now connected to the 360 using one of the fast wire links with a small PDP-11 digital controller together with a multi-crate CAMAC system; each alphanumeric terminal then requires only a single-width CAMAC module.

Any special equipment in the laboratory is interfaced through CAMAC.

5.4 Terminals outside the laboratory boundaries

All particle physics laboratories have users outside the boundaries of the laboratories and, in general, there is a need for these users to be able to use central computer facilities remotely without actually coming always to the computer installation. Naturally these needs vary, ranging from a fairly modest requirement for remote job entry to the very demanding fully-interactive graphics using large data bases. In principle there is nothing impossible in meeting the needs of remote users, although there are necessarily both technical and economic questions. Nevertheless, the problem is one which certainly involves data links and, as will be seen, it is also one where small computers can play an extremely important part.

The whole question of remote users has been studied at Daresbury, where an ambitious experiment has been set up to try to establish whether or not the most demanding of the user requirements, namely remote fully-interactive graphics, could be satisfied. The requirements for terminals outside the laboratory boundaries could be summarized as follows:

i) Standardization.
ii) Reliability - better than 1 bit error in $10^9$.

iii) High-speed - not less than $10^7$ bits/s.

iv) System transparency (FORTRAN or PL-1).

v) Many users.

vi) Arbitrary distance.

vii) Economical (??).

Most of the above requirements are the same as those for the domestic wire links, so it is clear that any external link should be regarded merely as an extension of the internal network. In this way the problem reduces to that of being able to transfer messages of arbitrary length across the laboratory boundaries at the required speed and with the required reliability. To do this an essential constraint is to use standard Post Office equipment outside the laboratory and, in particular, because of the requirements i), iii), iv) and vii) just listed, the best solution is to use a microwave channel. Of course, there is a considerable choice of microwave channel, ranging from a conventional video link to one or more PCM channels in a broad-band communications link. The problem of having an arbitrary distance is solved by having at least one regenerator at carrier frequency. Then, provided that messages can be satisfactorily communicated with one such regenerator, it should be possible to extend the distance just by adding regenerators; in this context, the regenerators are digitally passive in the sense that there is no knowledge at the regenerator that the signal is digitally coded.

The most difficult problem is that of deciding transmission strategy and communications protocol. There is no space here to discuss either the range of choice or the reasons for selecting any particular strategy. Suffice it to say that with distances of 50 km or more, and with transmission speeds of $10^7$ bits/s, any forward error correction scheme would prove to be much too inefficient. Instead it is necessary to adopt a continuous transmission strategy with a retransmission scheme for error correction. In order to do this it is essential to partition messages into blocks and buffer the blocks at both ends of the link until messages are correctly received
and re-assembled. For this function the small computer is ideally suited.

5.5 Implementation of a digital microwave link

In the communications experiment described, the aim was to interact with some extensive data-processing program (such as ASTERIX\textsuperscript{7}), an interactive graphical version of SUMX) by means of keyboard and light pen, but then to transmit the results back to the user via the external link over some distance which was any integral multiple of the basic hop length, in this case 70 kms. A block diagram of the total communications channel is shown in Fig. 13.

The microwave equipment used was a completely standard C-band (\textasciitilde 7 GHz) video link operating with about 1 W power and a baseband width of approximately 5 MHz. This equipment was not modified in any way, the only connections in addition to the power supply and antenna being two 75 \ \Omega coaxial connectors, one for input to the transmitter and the other the output from the receiver.

At the laboratory terminal, a DDP 516 was used as a buffer both for input and output. This small computer (without peripherals) was connected in the usual way to one port of the standard domestic high-speed link network. The function of the DDP 516 was to perform blocking, error detection, retransmission, message assembly and communications protocol.

The only new equipment necessary was a simple external interface between the Post Office installation and the standard Daresbury data network. In the event, this interface was implemented in the CAMAC format, with one single-width transmitter interface module and a similar module for receiver interface. In this interface there was encoding for 10\textsuperscript{7} bits/s and some error detection.

5.6 Performance of the link

One of the most important problems encountered in the microwave transmission of digital information is the correction of errors. In the particular case we are considering, the raw error rate observed was about 1 bit in 10\textsuperscript{8}. However, although the dominant form of error is random, there is a significant component of burst errors; there
Fig. 13 Block diagram of microwave communications channel
is, in addition, a component of catastrophic failure due to external physical effects (such as birds!!!), but we can ignore that question here. The question then is how to correct these errors until an acceptable reliability is obtained, and this is done by partitioning any given message into blocks, incorporating in each block the required number of identifier and check bits. Any errors in blocks are detected at the receiver, which eventually signals back to the transmitter the identification of all incorrectly received blocks; meanwhile transmission continues. At a suitable moment the error blocks are re-transmitted and the process continues until the complete and correct message is assembled at the receiver. The message can then be forwarded and the buffer spaces at both transmitter and receiver released.

In the transmission strategy just outlined the block size is a parameter although, as the block size increases, so do the overheads due to check and identifier characters. On the other hand the error probability is also a function of block size. Thus it is not surprising that there exist definite optimum conditions which minimize the time spent by the channel in transmitting non-message data. This is illustrated in Fig. 14 which shows the lost useful transmission time as a function of block size (horizontal axis) and random error probability (oblique axis) for a definite value of message length and overhead bits. In addition it can be shown that if there is a wide range of message lengths it is still possible to choose an optimum fixed block size, especially if the distribution of message lengths is known. In Fig. 15, the channel transmission efficiency is plotted as a function of block size (linear horizontal axis) and message length (logarithmic oblique axis), from which the existence of global optimal conditions can be deduced. One of the most important tasks of the small terminal computer in the microwave link system is the message partitioning and error correction.

The overall performance of the link is very encouraging. For a continuous transmission rate of $10^7$ bits/s, it is possible to attain a corrected error rate of better than 1 bit in $10^9$, so that the performance and reliability is at least as good as any computer peripher-
Fig. 14 Distribution of lost transmission time

Fig. 15 Efficiency of microwave channel
al. The link can be used with FORTRAN or PL-1, in this way obtaining complete system transparency to the user; who knows nothing about the very complex system between his input devices (high-level language programs, keyboard and light-pen) and the output (graphics or hard-copy). The performance, except for insignificant response-time delays, is distance independent, and experiments have been made with output results being re-transmitted 50-100 times around the microwave channel before being presented to the user. This correctly simulates the actual conditions for total transmission paths of say 5000 km, since each of the 100 hops included a regenerator at carrier frequency.
6. **CONCLUSION**

What I have tried to show is that the role of the small computer in particle physics applications is as a part of what is usually a fairly complex system. This system aspect should never be forgotten for the small computer, although essential, is after all only one component. Moreover it is a component not only in solving the applications problems of physicists, but also in the solution of many of the associated systems problems. In all this, it is particularly helpful to have modularity both in hardware and software, using the components to synthesize each new system; in this way, previous results and sub-systems can be rapidly incorporated into any new system. An equally important aspect of the modular system philosophy is that a given computational task can be distributed in such a way that each separate function can be implemented on that part of the system best matched to the needs. Distributed computing is also the best way to keep in step with the very rapidly changing situation both in the requirements of physicists and in the availability of new computer hardware and software from manufacturers.

We are at the beginning of the real development of distributed computing systems, made possible by the rapidly reducing costs of hardware, improved reliability and by the development of data links. It is difficult to predict what will be the correct balance of computational load in future situations between large and small processors. Nevertheless, despite the dynamically changing situation, we can be certain that the small computer will continue to play an essential part.
REFERENCES

1) H. Øveras, Dead-time losses in a buffered data recording system, CERN Preprint (April 1972) - submitted to Nucl. Instr. Meth.

2) CAMAC - A modular instrumentation system for data handling, Euratom Report EUR 4100e (March 1969).


THE OMEGA PROJECT: A CASE HISTORY IN THE DESIGN AND IMPLEMENTATION OF AN ON-LINE DATA ACQUISITION SYSTEM

R.D. Russell
CERN, Geneva, Switzerland

I. INTRODUCTION

In April 1967 a CERN working group was formed to study the feasibility of building a large magnet with wider solid-angle acceptance and larger useful volume within the magnetic field than any magnet previously built for a physics laboratory. With such a powerful tool experiments having highly complex interactions could be performed with detection of more secondary particles than previously possibly. After considerable investigation, this group published a proposal in May 1968 for building at CERN a large aperture spectrometer magnet with associated detector and data handling systems that should become operational in 1972. This proposal formed the basis of the Omega Project. Within these specifications, the project is given a very broad scope. It is not restricted to any single experiment, but is to be a general facility, similar to a bubble chamber, which can be used by many experiments having a wide range of requirements. It is expected to have a useable lifetime in excess of ten years, and therefore had to be designed to anticipate developments in both physics and technology; frequent modifications were to be expected.

As originally proposed in 1968, the primary tool for detecting and measuring physics events was to be a set of optical spark chambers located in the magnetic field. Initially all events were to be recorded on film, but with an anticipated advance in technology, the importance of the film readout was expected to decline as more and more experiments made exclusive use of filmless detector systems. So rapid has been the advance of detector and readout technology that the film system was abandoned entirely more than a year before the first scheduled operations, long before the magnet itself was completely constructed.

This illustrates one of the primary characteristics of the Omega Project's short history: its definitions and requirements are always
changing. Many changes are minor, but some, such as the replacement of the film readout system with plumbicon cameras, require considerable reevaluation of the entire project, especially the data handling aspects. We therefore cannot talk about the Omega Project as a static set of definitions and proposals. The Omega Project of 1972 is very different from that of 1970, and will undoubtedly be completely different in 1976, when operation of the 300 GEV machine begins. Fortunately this rapid evolution was anticipated in the original proposal, and throughout the course of the project flexibility and generality have been the primary design principles.

This paper is divided into two parts. The first deals with the historical evolution of the Omega computing system, and concentrates mainly on the problems that occurred and the decisions that had to be made during the past few years. The second part will deal with some of the more interesting technical accomplishments of the Omega data handling system. Since this project was intended to produce an advanced physics facility, the computing system includes many advance ideas and techniques that should be of general interest. The author is a visiting scientist at CERN who joined the Omega Project in the summer of 1971. Since he was not involved in the planning or development of Omega prior to that time, it must be understood that all comments and criticisms reflect an attempt to objectively appraise the project's evolution from the point of view of one who has seen the results of the decisions without participating in the debates that surrounded them.
II. THE EVOLUTION OF OMEGA

A. The Beginning

The original report distinguished three main functions of the data handling system:

1) On-line data acquisition and control
2) Data reduction (pattern recognition)
3) Data analysis (geometry and kinematics).

Although these three categories remain more or less valid today as production uses of the system, they are incomplete, since they omit all uses of the system as a development tool. There are at least two additional phases, which experience has shown to be equally or even more important than those originally mentioned. These are:

4) Computer system development (hardware and software)
5) Experimental system development (hardware and software).

Recognition of these last two functions as distinctly different uses of the computing system caused several of the more basic changes to the original computer system specifications. Until June of this year development activities have been the sole occupation of both the computer group and the physicists, since there was no beam before June, and hence no data to acquire, reduce, and analyse. Since the beginning of the project, use of the computer system has been essential for its own development and for the development of the experimental systems. This use of the computer as a development tool will not stop with the start of production data taking. If anything, it may become more important now that feedback from the production activities is available, and now that changes to equipment configurations and software must be made rapidly yet accurately in order to avoid loss of precious beam time.
B. The Initial Data Handling System

The Omega computer system specified in the working group's proposal of May 1968 is shown in Figure 1. It consists of a main computer and a small control computer, both on-line to the Omega experiment, and an HPD film scanner with associated control computer that is connected to the main computer. The main computer was specified as the primary data acquisition and data reduction computer. During filmless experiments, it would acquire data directly from the experiment's electronics and would record it onto magnetic tape after some rudimentary filtering. It would also perform some calculations on a small fraction of this data in order to give real-time feedback to the physicist about the quality of the data and the course of the experiment. These calculations would be simple consistency checks that could be done without floating-point hardware. A CRT display connected to the main computer would be used to display histograms of simply calculated parameters, such as spark frequency distributions.

It was calculated that a medium-sized computer with fast fixed-point operations could perform the production pattern recognition tasks more economically than the larger central installation computers. Therefore, when not on-line to the filmless experiments, the main computer would be executing pattern recognition programs on data received from the on-line HPD film scanner, and would be engaged in pattern recognition of recorded events from both sources. The CRT display would enable a human operator with a light pen to interact with the pattern recognition programs in order to recover events that could not be processed automatically.

The use of the main computer during the central part of the project's lifetime was projected to be roughly the following: 17% of the time for on-line filmless data acquisition; 10% of the time for film scanning with the on-line HPD; 40% of the time for pattern recognition of about five million filmless events; and 13% of the time for recognition of the one million film events. The remaining 20% of the available time was allotted for general development work. The
later stages in the processing chain (geometric reconstruction, kinematics, and statistical analysis and curve fitting) would be done on the more powerful central installation at CERN and in the computing centres of the member states.

The inclusion of a small control computer that would also be on-line to the filmless equipment was clearly a secondary consideration in the original proposal, where it was 'suggested ... for reasons of economy and safety', rather than being required as the absolute necessity it quickly proved to be. Its primary functions were for data acquisition and control during film-mode data taking, and as an aid in setting up and testing equipment. During filmless experiments, it would be by-passed by the normal data flow to the main computer, and could be allocated to control functions. It would be used for on-line data taking only if the main computer should fail.

Although it is easy to criticize this original proposal after four years experience with the system, I would nevertheless like to point out a few problems with it. Basically it underestimates both the value of Omega as a physics facility and the need for computing power to utilize this facility effectively during non-production operations. Although the report includes proposals for several experiments and indicates possibilities for Omega's use in future experiments, it did not seem to envision the overwhelming response of the physics community to the potential of the Omega facility. Within two years more than a dozen experiments had been proposed for Omega. Since most of these experiments involve complex triggering systems that require many months to set up and test, and since beam time in the Omega area is both limited and expensive, it was soon obvious that tasks originally expected to occur sequentially would have to be performed simultaneously in order to make maximum utilization of the facility. This brings us to the second shortcoming of the original proposal: Its lack of sufficient emphasis on the non-production aspects of the facility. Production data taking and analysis constitute
only a small fraction of an experiment's life-time. The majority of the time is spent in designing, developing, and testing both hardware and software. The original proposal emphasized production while underestimating the value of the data handling system as a tool for setting up and testing experimental equipment. This resulted in considerable expansion of the data handling system when the increased demand for access to Omega made it necessary to perform these activities in parallel with production tasks. The specifications did however stress flexibility as an important design principal, and adherence to this goal made it possible to expand the data handling system easily as the need arose.

C. The 2 System

The Omega Working Group's proposal was published in May 1968. Within a year the first significant changes occurred in the specifications. Due to the extremely rapid development of direct readout technology, the original estimates on the use of film readout devices were revised downward from one million events per year over a period of ten years to about 3000,000 events per year for only the first two years of Omega operation. Since film detectors were to be used so little, and since they would be eliminated so soon in the project's lifetime, it was decided not to build a special HPD film scanner for connection to the Omega computer, but rather to modify the existing CERN HPD-1 so that it would accept the Omega pictures. This HPD would remain connected to the central computer installation at CERN, so that the Omega computer would no longer be required to perform any film scanning.

With the increased emphasis on direct readout detectors, the rôle of the on-line computer immediately became more important, and the development of the entire filmless data acquisition system became more urgent than in the timetable of the original proposal. As shown in Figure 2, the configuration of November 1969 has eliminated not only the HPD but also the direct connection between the main
Omega computer and the on-line equipment. This was due to the realization that during set up and testing phases, a new experiment would need the facilities of an on-line computer almost continually, and this type of on-line work would inevitably cause the computer to 'hang-up' fairly frequently, due to hardware and software bugs. Therefore, production pattern recognition programs, which were to be run on the main Omega computer when production data acquisition was not in progress, could not be executed without undue interference unless these two computer activities were isolated into two separate machines, as was specified in the original proposal. However, once equipment has been interfaced to the on-line computer, which may require several months of testing to demonstrate that it functions properly, it is clearly not desirable to switch the interface to another computer for production, since this would not only require retesting everything, but would require development of duplicate software for the two computers (as well as duplicate hardware interfaces). The triggering and detection systems are extremely complex so that interfacing to one computer is difficult enough. There is no reason to make it more complicated by changing the interface between production and testing. Therefore it was decided that all acquisition hardware would remain connected to the on-line computer during production as well as testing, rather than being switched to the main computer as originally specified.

An added factor in this decision was that if a sufficiently powerful on-line computer were selected, this machine would be able to perform all the data acquisition and recording onto magnetic tape for experiments having data rates below about 20,000 coordinates per burst. During such experiments, which were expected to consume about 20% of the real-time in a year, the main computer would be used only to perform sampling calculations on the data sent via the link from the on-line machine. For experiments with higher data rates, which were expected to occur only 10% of the real-time, the main computer would record the data onto magnetic tape as it arrived over the data.
link from the on-line machine. During the remaining 70% of the year, the two computers would operate independently: The on-line machine being used for set up and testing, the main computer used for production pattern recognition and general development work.

D. Selecting a Computer

1. The requirements

Once the configuration shown in Figure 2 had been decided on, the choice of which computers to use had to be made. Without going into the details of the selection process (which lasted for many months), it is possible to delimit the major factors that were considered and their influence on the final decision.

Perhaps the biggest single factor was the data rates from the on-line experiments in filmless mode. The PS beam supplied to Omega consists of a 300 millisecond burst every two seconds. Although a maximum of 50-60 triggers per burst is possible with chambers having a five millisecond dead time, the highest average data rate was placed at about 30 triggers per burst. The maximum amount of data was estimated at 1500 16-bit parameters per trigger, which yields 45,000 coordinates per burst and an instantaneous transfer rate of 150,000 coordinates per second. Allowing for a buffer of about 25K of 32-bit words, a resident monitor, the acquisition programs, and the sample analysis program, a minimum memory size of 64K 32 or 36-bit words is seen to be necessary. The data rate also determines that tapes with a recording speed of at least 90 KC be attached to empty the buffer between bursts, but for reasons of future expansion, 120KC rates were preferable. It is interesting to note that at these rates, one tape would be filled every eight minutes, and the expected 50 million events per year would fill over 7000 tapes per year.

Since these figures were all estimates that depended on chamber dead time, amount of data acquired per trigger, and beam
characteristics, it was felt that all numbers (memory size, tape speed, etc.) should be exceeded by a comfortable margin, and that the system should be expandable to handle future increases in both size and speed.

Due to the high instantaneous data rates, it was specified that access to memory of both computers had to be one microsecond or less, in order to minimize read-out and transfer times. The main computer had to be capable of rapid fixed-point calculations in order to perform the on-line checking and the pattern recognition. Floating-point calculations were not expected to be necessary on-line, so that the floating-point hardware, while necessary, did not have to be exceedingly fast. The size of a memory word was placed at 32 or 36 bits, for numerical accuracy, but with operations available on the 16 or 18-bit half word, so that spark chamber coordinates (of 13 to 15 bits) could be handled without unpacking yet without wasting space. The availability of a large drum or disc, holding at least 20 million characters, was a necessity for system libraries, overlays, etc. Because this computer was to be heavily used for development purposes, especially during the initial period of Omega operations, it was essential that the machine be able to provide a facility for continuous, self-operated software development. This required that the manufacturer supply a general multi-programming supervisor as well as the standard software (FORTRAN, assembler, linking loader) and standard peripherals (card reader, line printer, card punch, console teletype).

The on-line computer was not required to be as powerful as the main machine, but did have to be capable of sustaining the high data rates needed by the readout equipment. A smaller word size (16 or 18 bits) and a smaller memory size (16K or 24K) were acceptable, although a fast access speed (one microsecond or less) was still felt to be essential, as were powerful, flexible I/O channels. It was important that the word size of the main computer
be a multiple of the on-line machine's word size in order to minimize problems with the link transfers and tape formats. The small machine also had to have a disc for program and monitor use, and 60 KC to 120 KC tape drives in order to record the data as it was acquired from experiments with slower data rates. The required software consisted of a FORTRAN compiler, an assembler, a linking loader, and a foreground/background supervisor.

2. The choice

Once the detailed specifications had been determined, evaluation of the proposals made by the manufacturers began. Some of the considerations used in making the final decision are summarized in Table 1. Benchmark tests, consisting of a number of FORTRAN programs similar to those expected to be used heavily in Omega, were run on all the proposed computers. This permitted a comparison of several important parameters: compilation speed, execution speed, size of the compiler, and size of the object code. In addition a number of subjective factors were obtained from these benchmarks, such as 'usability' of the system and general reliability of the hardware, software, and documentation. These qualitative considerations produced some unexpected discoveries. Many of the compilers contained bugs, so that legal FORTRAN programs did not work. Others ran but gave wrong answers due to hardware malfunctioning. One manufacturer never did succeed in getting the programs to run.

Besides the benchmark tests, the manufacturers' proposals were evaluated on the basis of 'how well' they seemed to meet the Omega specifications. Especially important was the expandability of the proposed configuration, since Omega was expected to grow and the first system was only a 'minimum' facility. Thus the ability to add additional core memory, disc memory, and I/O channels was of prime importance. A second important consideration was to determine what was already available and what was merely
proposed for some future time. Manufacturers often 'promise' software and hardware features which are only in the design or testing phase, so that any delay in their development would cause a corresponding delay in Omega. It was especially important that Omega not become the first user of a major piece of hardware or software, thereby becoming the one to find all the manufacturer's bugs.

Since the Omega computers were to be part of a general facility containing several critical components (such as beam-on time), it was important to have a reliable computing system. The hardware had to be dependable during production data taking that might go on 24 hours a day for several weeks, and the software had to be secure from interference between the production tasks and other users. Reliability of the proposed machines was determined from the benchmark tests and by making visits to other installations having a similar machine.

When the hardware does malfunction, as it inevitably must, it is extremely important to have competent technical people readily available to service the fault as quickly as possible. It was felt that maintenance personnel based in Geneva, if not on the CERN site, were essential for the Omega machine. Maintenance considerations also made it desirable to keep the number of non-standard peripherals (other than the experimental equipment) to a minimum. Tape drives or discs not supplied and supported by the computer manufacturer were felt to be potentially dangerous from the maintenance point of view, especially when these devices were so crucial to the production data taking.

Software maintenance is also an important consideration, and depends on the willingness of the company to correct any problems quickly and to make improvements to its software as dictated by Omega's needs. It was therefore important to have direct access to the responsible technical people within the company, rather than having to deal with public relations and marketing personnel.
for solutions to technical problems. A great deal about a company’s policy toward its users can be determined from its documentation: whether it exists at all, and if so, how accurate and up-to-date it is. Since it was anticipated that parts of the operating system of any computer decided upon would have to be rewritten to satisfy the unique needs of Omega, it was extremely important that the manufacturer make available to the Omega systems group all the source code for the operating system. Without this, necessary modifications to the system simply could not be made.

After all the machines were evaluated in detail, the final choice came down to two that met all criteria satisfactorily except the last one: price. One machine was significantly more expensive than the other, so the less expensive proposal was accepted. This resulted in the choice of the CII 10070, shown in Figure 3, as the main Omega computer. A similar selection and evaluation procedure resulted in the choice of the EMR 6130, shown in Figure 4, as the Omega on-line computer.

E. The 3 System

In early 1970 a decision was made to combine the Omega data handling system with that of another CERN project: the Split Field Magnet (SFM) Project. The split field magnet is a large magnet system located in intersection region 14 of the Intersecting Storage Rings (ISR) and designed such that the net effect of the magnetic field on the circulating proton beams is zero. Although development work had already begun on SFM by this time, it was not expected to become fully operational until the beginning of 1973. Both projects have a long expected lifetime and will generate a large amount of data that must be acquired by an on-line computer. Since SFM is also a general facility with data structured similar to the Omega filmless data, and since all the processing problems are expected to be similar, it was decided that the best approach was to combine the data handling
systems and to assign a single systems group the task of implementing the combined system.

The layout of the new system is shown in Figure 5. Since the main computer and the on-line computer for the Omega Project had already been selected and the orders placed, the simplest adequate solution was to order a second EMR 6130 to be on-line to the SFM facility, and to connect the main computer to both on-line EMR's via direct data links so that it could perform real-time sample analysis for either experiment separately, or both simultaneously.

The impact of this enlargement of the system was twofold: First, it made the EMR 6130's the critical bottleneck, since all data acquisition for two experimental facilities were now absolutely dependent on the capabilities of that computer: Second, it necessitated a more generalized link handling scheme in the main computer. Although the generalized link handler would have been designed anyway, the shift in emphasis from the CII 10070 to the EMR 6130 as the key computer for on-line operations was totally unanticipated by the original specifications. Whereas it was previously felt to be merely 'desirable' to allocate all on-line set up and testing to the EMR, it now became essential, since if the main computer were used for such tasks, malfunctioning of a test program at one facility would not only destroy a production analysis run (which is not too serious, since the data, safely on tape, could always be re-analysed), it would also destroy the sample analysis and/or data recording of an on-line experiment at the other facility. Clearly such interference with production data-taking must be eliminated, or at least held to an absolute minimum. As a result, the EMR 6130 became the sole computer to be used for set up and testing, as well as the most important machine during on-line acquisition (without it, an experiment could not proceed).
F. Designing the System

During most of 1970 the specifications for the software system were worked out in some detail. This included a detailed analysis of the dialogue for communicating via the computer to computer links, the integration of the link software into the computer operating systems, specification of a data acquisition and sample analysis system for a standalone EMR 6130, and specification of the CII data handling system and its expected use by both experimental facilities. Basic design principles were to make a flexible, modular system that could be easily modified or expanded in any direction, and to utilize existing software as much as possible, both that supplied by the manufacturer and that already available on other computers at CERN. This lead to the decisions that the CII 10070 was to operate with the SIRIS 7 system supplied by the manufacturer, and that graphics on both computers were to be done with the GD3 graphics package already developed on the CERN central installation.

It was also felt to be important to make the system as fail-safe as possible, both against equipment malfunctioning and against errors made by inexperienced operators. This was to be accomplished by building a library of well tested procedures for all basic tasks, such as link communication, display driving, and histogramming, and by building extensive monitoring and error recovery facilities into the software system. On the CII 10070 this required time-out monitors on all I/O activity, especially that involving the links, a revision to the existing FORTRAN library so that errors could be better detected and handled by the program, and a 'SPY' program for monitoring the progress of both user programs and the operating system.

Many other mundane details had to be worked out during this period, such as standardizing the tape formats for compatibility between all machines (including those at the central installation), and providing standard bookkeeping procedures for handling the documentation, tape storage, accounting, file editing, and program libraries.
On the hardware side, it was decided to use CAMAC for all interfacing between the Omega EMR 6130 and the counter and trigger systems. At this time, the spark chamber readout system was to be interfaced directly to the on-line machines, but in early 1971 it was decided to route this through the general CAMAC interface of the EMR 6130 as well. For the SFM experiment the chamber readout system still does not use CAMAC, although most other on-line hardware does. Tektronix T4002 display terminals were chosen for connection to the EMR 6130's, but since the CII 10070 was to drive at least two interactive display terminals having considerably greater power than the output-only displays on the EMR, more time was needed to study the options available. By early 1971 the decision was made to use two more Tektronix T4002 storage tube displays with keyboard and tracker-ball driven by a mini-computer linked to the CII 10070.

G. Expanding the System

By October 1970 the implications of the critical rôle played by the EMR 6130 in the on-line phase of all experiments at the Omega facility was fully realized. At this time, proposals for over a dozen experiments using Omega had been received, and due to the limited number of 14-day PS beam periods available to the Omega area (seven per year) and the length of time needed to set up and test the triggering systems for each experiment, it had become clear that in order to perform all these experiments between the time the facility would start receiving beam (June 1972) and the time Omega was scheduled to shut down operations in order to convert from optical to direct readout detectors, a new experiment would have to be ready to start production data taking every six weeks. Such a schedule could be met only if three experimental groups could have simultaneous access to the Omega facility: one to set up the triggering system; one to test it; and one to actually use it for production data taking. In this manner an experiment would be able to stay on the floor for a four-month period on the average. Since the EMR 6130 was an essential
part of each of these activities, it was felt that the three groups would have to have simultaneous access to this computer, especially during the beam-on time.

This represents a significant departure from the original specification of the EMR's functions as a single-user system, and in fact resembles more the requirements originally specified for the main CII computer. However, since one EMR had already arrived on site, and since the other was already ordered, it was not possible to consider acquiring a different machine for the on-line work.

In January the situation of the on-line computer became even more critical when it was decided to replace film entirely and begin Omega operations in 1972 with a new readout device called a plumbicon camera. This decision was prompted by experience with a prototype system at Rutherford Laboratory, and as the readout electronics for that system had been successfully interfaced to their computer via CAMAC, it was decided to interface the Omega plumbicon system to the EMR 6130 via CAMAC also. This in turn led to a decision that all on-line hardware used in Omega would be connected and controlled with CAMAC.

The plumbicon camera operates in a manner similar to a television camera: visual light from the spark chambers is focused onto a photosensitive surface where it is scanned by an electron beam in such a manner that each line in the raster scan corresponds to a single gap between planes of wires in the chambers. The position of any spark in that gap can be placed by the camera's logic with an accuracy better than 0.5 mm, and the digitized position coordinates are available for readout into the computer at the end of each scan line.

The Omega chamber layout would require eight of these cameras, six inside the magnet and two outside. Each camera would scan about 45 lines including the fiducials, and would be provided with eight scalers so that up to eight sparks could be digitized
per camera per line. Scanning a single line would require 68.3 microseconds, and reading the 64 scalers into the EMR would require at least 102 microseconds, so that the total time to scan and read out all 45 gaps would be 7.7 milliseconds.

Replacement of film cameras with plumbicons implied that Omega would have to be ready for on-line production data taking more than two years before the date originally specified for the first filmless experiment. In addition the initial load on the whole system, and on the on-line EMR 6130 in particular, would be much higher than estimated, and the need for the EMR to set up and test the new plumbicon system as well as the three independent trigger systems made this computer even more critical than before to the success of the entire Omega Project.

Many possible expansions of the data handling system to relieve the critical load on the on-line computer were discussed during early 1971, and it is worthwhile here to examine a few of these to see what the possibilities appeared to be, and what the real constraints proved to be. The trend clearly discernable in these discussions was 'downward' expansion of the system toward a symmetric arrangement of identical on-line 'satellites' that could be used independently of each other. The result of these discussions was the 4 system that is discussed in the next section.

The first possible direction for expansion was to design an EMR operating system that would be capable of supporting three users simultaneously. This was considered only briefly, and was quickly discarded because of the overwhelming technical problems involved. The system would have to be written entirely at CERN, and could not have been a multiprogramming system in any commonly understood sense of the term, simply because it appeared to be impossible to develop such a system on that computer, especially in the short time available before it was needed by the physics experiments. Enormous problems were involved if even a severely limited system were to be attempted, all due to the fact that a small computer such as the EMR 6130 was
not designed to function as a multiprogramming computer. It was too small (24K of 16-bit memory), lacked adequate peripherals, such as displays and teletypes that would be needed by each on-line user, had discs that were too slow for any reasonable roll in/roll out, and perhaps most importantly, lacked an adequate memory mapping or protection scheme that is essential in order to isolate simultaneous users from one another.

The problem of interfacing the EMR to more than one set of on-line equipment was even more formidable. Due to the channel configuration of the EMR and the design of the CAMAC interface, at most two CAMAC systems could be coupled to the computer at any one time. This made it impossible to permit each user his own independent CAMAC system, and required that a single CAMAC system with multi-user access be designed. Such a system is virtually impossible to make technically fail-safe from programming errors, spurious signals, channel blocking, etc. The only possible approach was to dictate rigid 'good practice' methods for attaching equipment and utilizing the interface during testing. But this assumes that both the hardware and the software are already at an advanced stage of testing, and makes the unavoidable hang-ups during set up catastrophic for the production data-taking experiment. Clearly the usefulness of the EMR for equipment set up and debugging was minimal, if not non-existent in such a system.

A second possibility that was also quickly discarded was the purchase of a second EMR 6130 for on-line use in Omega. The first EMR would be used exclusively by the single production experiment, and would be connected via a link to the CII for the on-line sample analysis, as was the case in earlier systems. The new EMR would be devoted exclusively to the set up and test users.

This suggestion, shown in Figure 6, has several advantages over the multi-programming approach. The production user is safely isolated from any interference with the test users, and the second EMR can be used as a backup should the first one fail during a run.
However, there are many technical problems, since a user would have to interface his CAMAC configuration to one machine for testing and to another, although of the same type, for production. Furthermore a 'multi-user' operating system for the second EMR would still have to be developed at CERN in order to handle more than one test user at a time. This system would be essentially the restricted one proposed previously and there would still be the problem of connecting two test users to a single CAMAC interface. Of more immediate concern was the fact that this approach would have been too expensive, since even a minimal configuration of only 16K memory, paper tape I/O and a disc (no card reader, line printer, tape units, or data link) exceeded the available Omega budget.

During this period discussions of a different though related nature were also going on in Omega concerning the interactive graphics facility for the main computer. By early 1971 it had become clear that a mini-computer connected via a data link to the CII 10070 could control a set of independent display consoles better than any special purpose display controller available on the market. Since this would be cheaper, more flexible, and offered much greater potential for future modification or expansion than a fixed controller, it was decided to purchase a mini-computer for the Omega graphics facility. Since a mini-computer offered such advantages for a graphics facility, it might also be useful for the on-line experiments during set up and testing, and this was the line of thought behind the next suggestion, shown in Figure 7.

This approach required that the system be expanded by the purchase of three mini-computers, one to drive the interactive displays, one for the test user and one for the set up user. One of the on-line machines was to be configured with 12K memory, paper tape input, no disc, and no FORTRAN capability, but connected to the CII 10070 via a data link to make up for its lack of power. The second on-line machine was to be more powerful, having 16K core, disc, and FORTRAN capabilities, so that it could be used independently of any other computer for advanced testing. The production user would still
remain on the EMR 6130. Each machine would be equipped with a CAMAC interface for the triggering system, which would include a Tektronix 611 storage tube display system. In addition, the EMR CAMAC would be connected to the plumbicon readout system and the beam parameter counters.

As described, this expansion suffered from two difficulties: the lack of a suitable FORTRAN on any existing small machine in the price range, and the difficulty of switching both hardware and software from the new machines to the EMR when going into production. A quick survey of existing small machines indicated that FORTRAN was just not an appropriate language for programming such computers. The compilers were usually slow, difficult to use (i.e. had poor diagnostics), and accepted only a rudimentary subset of the standard FORTRAN language. The object programs were usually large and slow, and quickly exceeded the capacity of the small computer's hardware. This meant that programming for set up and testing would have to be done in the assembly language, a tedious and error-prone process. Furthermore, everything would have to be reprogrammed for the EMR 6130 in order to go into production, a wasteful duplication of effort made more difficult by the scarcity of development time on the EMR 6130 due to its use in production data acquisition. A rapid and easy transfer from testing into production was further hampered by the necessity of reconnecting the on-line equipment to the EMR during the switch.

H. The 4 System

The last suggestion mentioned above was a step in the right direction, and after further discussions a concrete proposal was presented in April 1971 for the system shown in Figure 8. This consists of three identical on-line mini-computers without FORTRAN or disc, arranged so that any one of them could be linked to the EMR for production runs, while the other two could be linked simultaneously to the CII 10070 for set up and testing. These three mini-computers
would be the same type as the interactive display computer, in order to avoid introducing a fourth type of machine into the already complex system. Consideration of the requirements for the on-line configuration (8K memory, paper tape I/O, CAMAC interface, link to CII 10070) and the graphics facility (8K memory, disc, paper tape I/O, link to CII 10070) led to the choice of a PDP-11 as the Omega mini-computer.

This system seemed to resolve all the problems mentioned earlier. Since each test user had his own computer, with an independent CAMAC interface to his electronics, interference with other test users as well as with the production user was virtually eliminated. The transition from test stage into production was now considerably simplified since during production the triggering system and the readout programs would remain on the PDP-11 where they were tested and debugged. Only the plumbicon and beam parameter readout system was connected directly to the EMR, and since this facility was expected to change very little from experiment to experiment, a general program for the EMR could be written to read the chamber and beam data, merge it with the trigger data received from the PDP-11 that was connected at the time, and record it onto tape. Thus the experiment-independent tasks of acquiring chamber data and recording it onto tape are clearly separated from the experiment-dependent tasks of acquiring and checking trigger data.

The big advantage of this system is its symmetry, a factor that took on added significance at this stage in the development of Omega when it was not known which of the first three scheduled experiments would be ready for production data taking first. Therefore it was essential that all users have the same set of facilities available to them so that no one group would have an advantage over the others due to the fact that its equipment was attached to the EMR for testing and the others' was not, or because its test programs were written for the EMR and the others' were not.
The problem with this proposal was that it was too ambitious to have completed in its entirety by the fall of 1971, when first testing of on-line equipment was scheduled to begin. There were several aspects to this problem. First, there were three on-line PDP-11 computers that had not been anticipated a few months before, so that all the software for these machines had to be designed and implemented. To further complicate matters, the machines themselves would not arrive at CERN until mid-summer, and the CAMAC interface, to be built by the manufacturer, was not expected before fall.

Second, the number of links in the system had grown from one between a single EMR 6130 and the CII 10070, to six between the CII, two EMR's and four PDP-11's. All the link hardware and software was being designed and built at CERN, with the first link (between the EMR and the CII) expected to be ready for testing in August, 1971. Now a new link interface to the PDP-11 had to be designed, new software written for the PDP-11, and the entire link scheme re-evaluated for consistency, since it was considered highly desirable that all links be as nearly identical as possible in hardware and software. The task of producing six sets of link hardware in such a short time greatly exceeded the capacity of the CERN electronics shops, which after all are not set up for assembly-line production of such devices.

Third, initial program development for the mini-computer would be extremely difficult without the links to the CII 10070 or some alternative. In the final system programs to set up and test trigger systems could be largely written in FORTRAN and run on the CII 10070, so that only a minimal amount of assembly language coding would be required on the mini-computer. Without the links, everything had to be done on the mini's, and since they were to have only 8K memory and no disc, they could not support a FORTRAN compiler. Therefore all the initial programming would have to be done in assembly language unless a compiler for a better language could be written to run on the CII 10070 and produce code for the mini-computer.
Finally, the enlarged scope of this system required considerably more manpower than originally estimated, and competent people cannot be obtained and integrated into a development group overnight.

I. The 5 System

Because of the preceding considerations, it was decided by July that, although the 4 system was highly desirable and should be implemented as a long term objective, a simplified, interim system should be developed for the initial period of Omega testing and production. The system is shown in Figure 9. It consists of all the computers of the large system, but with links only between the CII and the two EMR 6130's (the links specified a year earlier), and between the CII and the display PDP-11. Since all trigger data is read out by the PDP-11's, but must be merged with chamber data and recorded onto tape by the EMR, a uni-directional CAMAC to CAMAC interface has been added between the EMR 6130 and the PDP-11 of the production user. This interface is simply a buffer into which the PDP-11 deposits the trigger data it has acquired from the experiment's on-line equipment. The buffer is then emptied by the EMR 6130 after all the chamber data and beam parameters have been read out from the EMR's CAMAC. Other than this, no communication between the two machines is possible, and there is no way for the PDP-11 to delay the EMR 6130. If the PDP-11 does not fill the buffer in time, an error condition is detected on the EMR, a message is printed to the operator, and the event is abandoned. Such a scheme is far from fail-safe, but offered the advantage of requiring almost no changes to the existing EMR data acquisition software, of creating no danger of destroying the chamber data or inhibiting the tape recording process, and of being an extremely simple method of interfacing the two computers within a short time.

With this configuration, all the programs for set up, testing, and production use of the trigger systems must be run on
the PDP-11, and since a FORTRAN language is not available on such small configurations, it became obvious that some sort of programming assistance would have to be given to the users. Therefore a library of standard procedures and an on-line user's monitor were specified and developed to handle the more routine data acquisition functions, and to provide drivers for the Tektronix 611 displays and the standard I/O peripherals. As a more direct aid to the programming difficulties of the user it was suggested that a FORTRAN compiler be written to run on the CII 10070 but to produce object code for execution on the PDP-11. This met several objections: first, that the compiler probably could not be completed before it was needed by test users in the fall, even if only a subset of FORTRAN were implemented; and second, that FORTRAN was not an appropriate language for programming the PDP-11, especially for on-line, real-time applications. Instead it was proposed to design and implement an intermediate-level language, called PL-11, which would combine the syntactic form and programming style of a high-level, ALGOL-like language with the run-time efficiency and closeness to the machine hardware of the assembler. Such a tool would make it possible to easily program the PDP-11's without a data link to the CII and without a FORTRAN compiler for the mini-computer.

J. The Past Year

The idea of implementing the 5 system as an interim version of the 4 system seems to have satisfied all the necessary requirements for Omega, and since its final approval in August 1971 no major revisions have occurred. The past year has been spent implementing the various aspects of the proposal.

The first PDP-11 and the Omega EMR 6130 were delivered in July 1971, the second PDP-11 in August. During that summer tests were begun on the plumbicon system connected via CAMAC to the EMR 6130. In the fall, work was begun on the software for the PDP-11/CAMAC interface, including a set of drivers for the 611 storage tube displays, and on a set of FORTRAN-callable histogramming routines
that would comprise a compatible histogramming package on all three computers. By this time the first version of the PDP-11 on-line monitor, which was begun in the Spring, became operational, and by November the first version of the PL-11 compiler was available on the CII 10070 for general use.

The CAMAC interface for the PDP-11, the first of its kind built by the manufacturer, arrived in December, but it took several months of debugging before it finally worked properly. Testing of the first data link between the CII 10070 and the EMR 6130 began in December, and was completed by March, when testing of the link between the CII and the display PDP-11 began. By this time the last two PDP-11's had arrived, and all were in use by experimental groups for equipment set up and testing. By April the link between the CII and PDP was reliable enough to permit use of the interactive display terminals by the physicists. During this period a great deal of development effort also went into the applications programs on all three machines, particularly 'ROMEO' (Reconstruction of Omega Events Off-line), which is a general program written entirely in FORTRAN on the CII to perform pattern recognition and geometric reconstruction of a large class of events expected in Omega.

In June the first beam appeared briefly in the Omega facility and has been available for three and four-day periods every several weeks since then. These periods have been utilized at one time or another by all three experiments, both for testing and for data taking with the PDP-EMR system. Although the CII is not on-line to the acquisition hardware directly, it is still essential to the on-line operations, since development and correction of all PDP-11 programs must be done with the PL-11 compiler on the CII 10070, and data recorded onto tape by the EMR is analysed on the CII during the run in order to get feedback on the quality of the data. In addition, the direct link to the EMR has been used by one group in a preliminary manner to transfer sample events to the CII where they were processed by the pattern recognition programs and the results returned via the
link to the EMR for printing. By the end of September geometric
reconstruction and simple kinematics programs will also be utilized
on the CII in order to allow useful physics histograms to be accumulated
in real-time on the sample of events that are selected and sent over
the link from the user program in the EMR.

K. The Future

Now that data taking has begun, a great deal of development
must still be done on the current system to bring it up to the spec-
ifications of the full 4 system. Although tested and working, the
data links that do exist have not really reached their full potential
and are as yet little used. Work is already underway to integrate
the CII/EMR link into a sophisticated sample analysis system with
elaborate histogramming, display, and control facilities, and to
integrate the CII/PDP link into a powerful debugging tool for use
during set up and testing.

On the CII 10070 a roll in/roll out system is being
developed to allow many real-time users to have access to the power
of that machine without degrading its performance. At the same time
a file handling subsystem is being implemented for use with the data
links to give the on-line computers remote access to the full range
of the CII facilities. On the EMR 6130, the first version of the
data acquisition and sample analysis system is being upgraded to make
it more flexible. This requires elimination of the many deficiencies
in the manufacturer's software system, which lacks a good overlay
facility, tape labelling, and user libraries.

L. An Evaluation

That completes the history of the Omega data handling
system from its inception in 1968 to the present. Except for the
past two months, it is a history of change and development, with
no 'production' physics. The major revisions required in the original
specifications demonstrate how carefully the 'development' phase of
such a large system must be considered, and prove the absolute necessity of the computer as a development tool for on-line physics experiments: it is as essential for setting up and testing equipment as an oscilloscope. Our experience has shown that designers of future on-line data handling systems must consider these two development functions in addition to the production functions of their systems if they are to avoid major revisions at a later stage. The items listed in Table 2 comprise the functional aspects which have evolved in Omega, and the most important consideration in the final system is its method of handling all of these functions simultaneously for a set of independent experiments.

The Omega data handling system was operational when the first beam appeared, a remarkable achievement in view of its complexities and the revisions which occurred. Furthermore, three months of operations have demonstrated that it is a useful system, although the rigorous test will not come until January 1973, when construction of the Omega magnet will be completed and full production data taking is scheduled to begin. This success was due to many factors. Perhaps most important were the basic design principles of generality and flexibility, but it must not be overlooked that all the Omega decisions were tempered by a realistic appraisal of existing options and constraints. The idea of starting with simple yet adequate interim solutions instead of attempting the full final system straight-off not only ensured successful attainment of a working system, it also made the many redesigns easier to absorb. A simple first step is often the most flexible method of finding out where the real design problems lie without endangering the success of the project. Once the simple base is made, more advanced stages are easily built upon it.

The original report anticipated frequent change in the definition of Omega, and experience has borne this out. Although there were defects in the original proposals they were detected and corrected at an early stage, indicating perhaps that flexibility is more important than having correct initial designs. In a broader sense the
the evolution of Omega reflects a growing trend toward multiple computer systems, and in particular toward a system in which large numbers of mini-computers are used for all interfacing to the external world. The result is a powerful, usable on-line data handling facility.
III. TECHNICAL ASPECTS OF THE OMEGA DATA HANDLING SYSTEM

A. Introduction

The most obvious aspect of the history of the Omega data handling system is its evolution into an integrated hierarchy of computers: one medium-sized, two small, and four mini. As discussed in Part Two, this system appears to offer the best solution to many problems that confront the designers of an on-line data acquisition system.

There are many interesting technical problems involved in designing and implementing such a complex system, and this part of the paper will present some of these in detail. We feel that many of the features of Omega have general application and should be carefully considered in the design of future systems.

B. The Data Links

1. The Concept

The most obvious first topic for discussion is the data links that join the individual computers. As shown in figure 8, the final system contains six links, and the interim system shown in figure 9 contains three. All the hardware and software for these links, and the interfaces to all three types of Omega computers, were designed and built at CERN. Since these links form such an important part of the data handling system, a great deal of thought and effort went into their design, implementation, and testing. A particularly important goal was to ensure that the operating system would be safe from errors in the users' FORTRAN programs, errors due to hardware failure, and errors in the computer at the other end of the link (such as its not being connected), and that any conflicts that might develop during the dialogue between machines would be automatically resolved. In addition, the link software was to be as close as possible to the basic functions of the link hardware in order to give a maximum of flexibility.
It would then be possible to build more complex systems, such as a file handling system, from the basic link primitives.

For purposes of flexibility, the link software was designed to permit any number of programs in one computer to communicate with any number of programs in the other computer over the same link. It was also planned to allow a program in one computer to communicate over different data links to several other computers, but because of implementation difficulties this is now permitted only on the CII. Finally, it was considered important that the user interface be as nearly identical as possible on all computers. This would make it easier for the user, since only one set of software routines need be learned, and also preserves the basic symmetry of the link hardware as seen from either computer. It does however require greater care in dealing with error conditions and transmission conflicts, such as occur when the two computers both wish to initiate a communication at the same time.

2. The user interface

The FORTRAN user/data link interface for the CII and EMR is shown in table 3, along with the equivalent PDP-11 interface. The reason for the differences in the parameters on the two machines is a simple but important consideration: multi-programming. On the CII, the general multiprogramming facilities provided by the operating system can be used to full advantage by the link system, whereas on the EMR no such facilities are available. The CII operating system is organized to handle multiple tasks, with all the necessary mechanisms to handle task activation, task blocking and task switching. It is therefore easy to have many simultaneous, independent link users. On the mono-programming system of the EMR, however, the concept of tasks and task switching does not exist, so that it is very difficult to share control of the processor between several users. To do so would essentially require the operating system to be rewritten for multi-programming. Therefore, although
the user routines are functionally equivalent on both computers, the actual details of their implementation and use must necessarily be different. Only the more general CII interface will be described here.

A program wishing to use a link must first call "connect link" (CNCTLK) in order to make itself known to the link interface. This sets up a buffer to receive transmissions over this link to this program, and a "call-cell" that will be used to identify the program at the other end of this link that is currently communicating with the program. A program can be simultaneously connected to any number of links, but at any one time it can receive transmissions from only one program per link (i.e., the one identified in the call-cell). The "disconnect link" (DNTLK) routine is called to terminate a program's use of the link, thereby releasing the buffer space and the call-cell. Typically a user program will call CNCTLK when it first begins execution and DNTLK just before it terminates execution. Conceptually, CNCTLK is equivalent to the standard "open file" operation and DNTLK to the "close file" operation.

In order to initiate a communication with some program B in the other computer, the user program A must first call "write link" (WRITLK) with parameters to indicate the location and amount of data, and the name of the program (i.e. B) in the other computer. The receiving program's identification (B) is placed into program A's call-cell for this link, thereby establishing B as A's current partner on this link and ensuring that the only future transmissions to program A that will be "accepted" on this link by the interface will be those sent by program B. Similarly, this transmission will be refused by the interface in the receiving computer unless program B's call-cell for this link is either clear or contains program A's identification (as set by a previous transmission between A and B).
The data sent to program B can be interpreted either as actual data or as status information from which it can determine the nature of future communications desired by program A (such as "send more data", etc.). If program A expects a reply from program B, it must call "read link" (READLK) to inform the interface that it is waiting for a transmission on a particular link. Program A is put into the waiting state until a transmission to program A is received from program B, at which point the data area specified in the call is filled with the data received over the link and program A is reactivated. The data received can only be from program B (since B is the program name in A's call-cell as the result of the previous WRITLK), and should be the data requested by the previous WRITLK from program A. However, it is the task of communication partners to ensure this, which can be easily done by keeping user status information in the first few words of each block transmitted. The "read multiple" (RDMULK) operation is used identically to the READLK except that a transmission to program A over any link will be accepted by RDMULK.

The interface routine "idle link" (IDLELK) is called by program A when it has finished all the communication it expects over the links it is connected to, and is prepared to wait indefinitely until a new transmission is initiated from an unspecified program in another computer. The interface clears all the call-cells for program A (one for each link to which A is connected), and then enters program A into the waiting state with no specified time limit. On the CII, a future system will make it possible for programs in this state to be rolled out onto the disk, so that they will not occupy valuable memory space. Such a facility is required for general purpose file handling programs in the CII which may be requested at any time by programs in the remote computer and therefore must always be available for use, although they clearly should not occupy memory between uses. When a transmission to program A is received from some program C in another computer,
the system reactivates A, rolling it back into memory if necessary, and establishes C as A's communications partner on the appropriate link by putting the name C into A's call-cell for that link. At this point, program A must call READLK (or RDMULK) in order to receive the data sent from the other computer.

3. The link dialogue

The interface just described obviously requires a software package to perform the link transmissions, although the link users are never aware of the details of the actual dialogue.

Each link communication consists of three parts, as shown in figure 10. The first phase is simply a "wake-up" exchange of interrupts that inform the initiating computer whether or not the receiving computer is "alive", and that indicate to the responding computer that it is wanted in a communication by some other computer.

The second phase consists of a write by the initiating computer and a read by the responding computer in order to exchange a standard block of status information between the two software interfaces. This information includes the program identification of the sending and receiving programs, and the number of bytes to be transmitted. The end of block interrupt also includes the status of the transmission as seen by the initiating computer. The receiving computer must digest this information to determine whether or not it can accept the transmission. There are several possibilities: (1) the requested program is connected and the link buffer is empty, ok to proceed; (2) the program is connected but the link buffer is full, try again later; (3) the program is not connected, illegal operation; (4) data block too big to handle, illegal operation; (5) error on transmission of the status block, try again. This response is put into a status register by the requesting computer and a return interrupt is sent to the initiating computer, which
then reads the status and decides what to do next.

If the responding computer has indicated that it can accept the data, the initiating computer will then issue a write and the responding computer a read in order to transmit the user's data block. Again the end of block interrupt is sent from the initiating computer along with information on the status of its transmission. The responding computer will read this status and then send back an interrupt to the initiating computer along with status information to indicate whether or not the entire transmission was successful, and if not, whether to try again or break off.

The software has been designed so that the initiating computer will only write, never read, and the receiving computer will always attempt to read more bytes of data than it knows the initiating computer will be writing. In this manner, it is guaranteed that the write will always terminate and the read will never terminate unless there is a hardware failure. This permits the initiating computer to receive status information from the link hardware on the quality of the transmission (parity errors, etc.) and send this, along with instructions on what to do next, to the responding computer as part of the end of block interrupt. When the responding computer receives the end of block interrupt, it knows there are no more data words to be transmitted, and can terminate the "hanging" read. In this manner both computers are kept completely synchronized, and the state of each machine is well known at all times to the communications partner. This ensures an orderly dialogue, especially in the case of conflicts, errors, or delays caused by lack of buffer space.

While this dialogue may appear a bit complicated, it does enable the software interface to easily detect conflicts and all errors, thus guaranteeing the integrity of the link in all communications and removing this essential task from the user. It also has been designed to fit conveniently into the CII multiprogramming system, which can be used to buffer all transmissions received over
the links and to wake up the receiving program after the sequence has been completed. If necessary, the receiving program will be first rolled in off the disk. The interface design has made testing of both hardware and software quite straightforward, due to the frequent exchanges of status and the close synchronization of the two computers by means of interrupts. This should also simplify detection of hardware failures during normal operations.

4. The link hardware

Finally, a few words about the link hardware. The links are full duplex serial transmission channels capable of handling rates up to one million 8-bit characters per second. Each interface contains a small buffer for both data and status, so that high data rates can be maintained without loss of data when receiving a transmission over a 1600-meter cable. In practice channel operation is limited to 400,000 bytes/second, the maximum data rate that can be handled by the CII. The CII 10070 interface is in the form of a device controller attached to one of the standard multiplexed I/O processors (MIOP's). The EMR interface is connected to the standard telemetry data channel (TDC) which is capable of handling data rates of 1,320,000 bytes/second in units of 16-bit words.

The PDP-11 interface has been built to accept from the unibus commands to initialize internal registers for a direct memory access transfer of a block of data. This approach eliminates the programmed word by word transfer that is characteristic of most I/O on the PDP, thereby allowing the transmission to proceed independently of the central processor and at rates approaching the memory access speed.

An important design principle was to make the data link as symmetric as possible. This not only made the system conceptually cleaner and easier to program, it also made hardware debugging
simpler and enabled the designers to build a sophisticated remote end simulator that could be used on-line and off-line to test link drivers. In addition, rather simple simulators of the CII MIOP and the EMR TDC were also constructed for preliminary testing purposes.

C. Omega Graphics

It is clear from the historical sequence of proposed Omega configurations that CRT displays have always been a prominent component, as they should be in any modern data handling system. In the current Omega system, each computer is connected to at least one CRT display.

Each of the three on-line PDP-11's is connected via CAMAC to a Tektronix 611 storage tube display. By making calls to a set of software routines provided in the Omega PDP-11 on-line monitor (see table 4), the programmer can conveniently perform the basic operations of point plotting, character generation, vector drawing, and erasing the storage screen.

Each EMR 6130 is interfaced directly to a T4002 storage tube display. While not as elaborate as the facility provided to CII 10070 users, this device forms an integral part of the sample analysis system on the EMR, and is heavily used in all phases of EMR operations. Associated with each of the on-line PDP-11's, but not connected to the PDP-11, is a second Tektronix 611 display which duplicates identically the picture displayed on the T4002 connected to the Omega EMR. This provides the PDP-11 user with an inexpensive method of monitoring the spark chambers and beam detectors, which are connected only to the EMR 6130, since no hardware link is necessary between the two computers.

A single large refreshed display with a 4K memory buffer of its own is now being tested on a PDP-11 for installation in the plumbicon CAMAC system of the EMR. This display will provide a continually updated picture of the spark coordinates superimposed on the current chamber set up independently of any computer. This is
invaluable for testing and monitoring the performance of the chambers, the triggering systems, and the plumbicon cameras.

The fourth PDP-11 is devoted exclusively to graphics, and is connected through a special interface to several interactive display consoles, each having a Tektronix T4002 storage tube display with keyboard input, scratch-pad memory, and tracker-ball. Currently two display consoles have been installed, with two more expected in the near future. This system has been designed so that a user at one of the display consoles can interact directly with a program executing on the CII 10070. The PDP-11 is essentially "transparent" to him, since all his programming for the displays is done in FORTRAN on the CII 10070 using an interactive graphics package called GD3. This package was originally developed for the CERN central computer installation, but has been rewritten for the CII 10070 so that to the FORTRAN user it looks identical to the original GD3. This enables FORTRAN programs that utilize graphics to be transported easily between the Omega machine and the central computers, an important requirement for efficient development and production use of the pattern recognition and geometry programs.

The GD3 subroutines on the CII 10070 are written as if they were connected to a standard display controller, but instead of being routed via a data channel to a hardware device, the display commands generated by GD3 are sent via the data link to programs in the display PDP-11, where software tasks interpret the commands and generate the I/O instructions necessary to drive the T4002 displays.

The decision to drive the display consoles with a mini-computer and link system rather than a hardware display controller was made for several important reasons. Clearly a mini-computer is far more flexible than any conceivable hardwired controller, offering virtually unlimited potential for modification and expansion of the display facility. Of course, a computer can be used for other functions when not in use by the display facility. By performing the
actual picture generation and keyboard input on the PDP-11, a faster response can be given to user button pushes than seemed possible with software on the CII running in a multi-programming environment. Furthermore, a display controller connected directly to the CII would have been a new type of device for the CII software to handle, with all the attendant development effort, but since the link system was being installed for the EMR's and the on-line PDP-11's, no additional development was required in the CII, and in fact the display system provided a convenient vehicle for testing the link hardware itself. As if these factors were not sufficient, the minicomputer approach was also cheaper than any device controller of comparable power.

Although the display PDP-11 functions only as a very "smart" device controller to users of GD3, this machine has been provided with basic but fairly general multi-programming functions in order to facilitate the programming and debugging of the display package, and to provide a system that can be easily expanded, not only with more T4002 displays, but possibly also with other I/O units, such as teletypes or other types of display consoles. This system provides general mechanisms for reserving and releasing resources, for handling task switching, for managing memory buffers, and for queuing disc and link transfers. The multi-programming approach also made implementation of the link interface simpler, since the existing CII link software could be used as a working model.

D. Histogramming Facilities

One of the most heavily used features of the Omega software system is the histogramming package. Because graphic feedback is essential during all phases of on-line work, an on-line data handling system must provide comprehensive histogramming facilities that can be utilized easily by the programmers and by the on-line operators.
In Omega, the histogram package has been integrated with the display system and link systems in a very powerful manner. It can be used either on-line or off-line, and once again the design was guided by the principles of generality and flexibility coupled with the desire to provide compatible facilities on all three types of computers. Clearly it would be impossible to attempt to provide on the PDP-11 all the features that can be provided on the CII 10070. However, the same basic kernel is found in all three machines, and to the extent it could be implemented easily, the use of this kernel and the results it produces are identical on all machines. Table 5 lists the one-dimensional histogram routines which were available in the first version of the Omega software system.

In order to define a histogram the user program must first call HBOOK1. This is done typically at the start of an experimental run. The system will then reserve the necessary bin space, reset it to zero, and set up the appropriate information for accumulating the data and making it available for output. Each time a value is computed that must be entered into the histogram, HFILL1 is called. This is done typically once per event for each histogram being accumulated. The system will scale the value and update the corresponding bin by one. Underflow and overflow values are also detected. At any time, HSTDOL can be called to print the histogram or display it on the CRT, depending on the parameters. A specified histogram can be reset to zero by calling HZERO, and the routine HFREE cancels a histogram, thereby freeing the space reserved for it. HMENU produces a list of all the histograms that have been booked and not yet cancelled.

Two-dimensional equivalents of HBOOK1, HFILL1 and HSTDOL are also available to the user. In order to conserve buffer space it is possible for the user to define for two-dimensional histograms the maximum number of bits which should be used to hold the bin values. Although this requires considerably more work in the histogram package to update and display a histogram, it makes possible
the accumulation of fairly large two-dimensional histograms on a mini-computer. It is also possible to obtain a display of the two-dimensional histograms in either tabular or scatter-plot form.

In all computers histograms are represented in the same internal format in order to make it easy to accumulate a histogram in one computer, then send it over the link for display or printing on another computer. In the near future additional facilities will be provided to allow the user to have on-line control of the histogramming package without the need to recompile his program. System commands will enable the user to display, define and cancel histograms from the teletype while his program is running.

E. Consequences of the Multi-Computer Hierarchy

An important consideration in the Omega data handling system is the allocation of various tasks to the component computers. In many respects the hierarchy of computers in Omega represents an ideal situation; a basic model which future on-line systems would do well to follow. This is especially true of the relationship between the CII and PDP-ll's. The CII is a fairly large machine, with an advanced multiprogramming system and a full range of file handling facilities. It was therefore decided that this machine should be used for all PDP-ll program development. The PDP-ll is designed to handle a wide variety of non-standard I/O devices in a very convenient manner. Therefore, all on-line equipment, including displays, should be attached to this machine. (Equipment is also attached to the EMR, but as a permanent experiment-independent facility that changes little, if ever, over time).

These were important decisions, and experience has shown them to be correct. First of all, since no program development is to be done on the PDP-ll, a minimum configuration suited solely to the on-line requirements is perfectly adequate. Such a configuration does not include card reader, line printer or disc, all of which
are essential on a usable program development machine. 8K of memory, teletype, and high-speed paper tape reader plus display units comprise a system that is perfectly adequate for setting up and testing on-line equipment, as well as for providing an on-line monitor of the trigger system during production data taking. These systems are very cheap, which implies that there can be many of them. In Omega, each experimental group has its own PDP-11, which means that each group can be completely independent of the other groups while setting up and debugging its trigger system. This not only increases the efficiency of the testing effort, since all problems with mutual interference (at the hardware and personnel level) are eliminated, but also provides each group with equal facilities and access to those facilities that is scheduled according to the particular needs of the individual group. (That is, one group is not forced to come in at 2 a.m. in order to be able to use the computer to test its hardware).

Concentration of the mini-computer program development on a large central machine clearly provides the mini-computer programmer with a better software development tool than could be provided with a set of identical small or medium-sized computers. The large computer has bigger and faster discs, faster line printers and card readers, more tape units and more memory than could possibly be provided at comparable cost for a set of independent smaller machines. These features plus an advanced multiprogramming operating system make the CII a powerful development tool for many simultaneous users. Furthermore, the data processing nature of program development, such as file editing, compiling and link-editing, is very different from the nature of tasks that test on-line equipment, which implies that interference between the simultaneous users of the central machine can be essentially eliminated by a good operating system, a task that is virtually impossible on the on-line computers.

The software development system of the large machine is not only better but can also be achieved with less cost than it could be for a set of smaller machines. This is due largely to the fact
that a properly chosen large machine will be delivered from the
manufacturer with a large complement of working software. Thus
many of the development tools already exist, and these can be read-
ily used to develop additional development tools in a boot-strapping
process. In the Omega system, a set of macro-instructions within the
CII METASYM assembler produced a PAL-ll "assembler" for the CII in
a matter of weeks, and the PL-ll compiler for the CII was written in
FORTRAN in a matter of months. Both of these compilers produce relo-
catable object modules in the CII object language format, which
implies that all the library and link-editing facilities of the
existing CII operating system were immediately available to the PDP-ll
programmer "for free". Any improvements in the CII software would
automatically be improvements for the PDP-ll users as well, thereby
eliminating any wasteful duplication of effort by systems programmers
to develop similar facilities on different machines. (One should not
overlook the amount of effort necessary to develop adequate data
processing facilities, even in a physics laboratory with an ostensibly
complete software system provided by the manufacturer).

The preceding considerations are valid whether or not the
two computers are physically connected. They need only be in physical
proximity to one another to be effective. However, once a reliable
data link does exist between the machines, the whole becomes truly
more than the sum of its parts. The central machine has gained a
very powerful yet relatively cheap peripheral, and the mini-machine
is suddenly endowed with direct access to processing power far beyond
its normal capacity.

Consider first the role of the PDP-ll as a CII peripheral.
Its importance as a controller, tester and read-out box for the on-
line trigger systems has already been discussed and is beyond question.
We have also mentioned the importance of the mini-computer in the
interactive graphics facility. The use of a mini-computer to drive
a set of displays gave more flexibility and better user response at
lower cost than was possible with any of the special purpose display
controllers considered. And of course, the mini-computer can be used for other tasks when the displays are not in use.

But perhaps the most interesting aspect of the mini-computer as a peripheral to the large machine is one that is now being developed in Omega: the mini-computer as a remote I/O station for the large machine. In particular, a system is being designed for Omega whereby a user at the PDP-11 teletype will be able to manipulate and edit files on the CII disc, submit jobs for execution on the CII and receive the results at his PDP-11 terminal. This benefits both the PDP-11 users and the CII users simultaneously.

From the point of view of the PDP-11, a link to the CII has many attractions. As mentioned in the last point above, direct access text-editing and file manipulation facilities on the CII will greatly enhance the already existing PDP-11 program development features on that machine (PL-11 compiler, link-editor, etc.). In addition, direct loading of the PDP-11 memory from the CII disc eliminates the need for paper tape to transfer programs, not only reducing the time the PDP-11 is idle during program modifications, but also enabling the PDP-11 to be located at a considerably greater distance from the CII than would be possible in the current setup. The link of course allows any PDP-11 program to communicate directly with CII programs, thereby making it possible to write large, sophisticated analysis programs in FORTRAN for the CII that obtain raw data directly from the PDP and return the results for display on the PDP. This may be of great use in the advanced testing stages of complex trigger systems. In addition, programs executing on the PDP-11 can be overlayed from the CII disc via the link, eliminating the need for a disc on the PDP-11 to perform this function. The discs on the CII are already being used for storage of PDP-11 program libraries, but the links now make it possible for programs in separate PDP-11's to communicate dynamically via the CII discs. The possibilities seem to be endless.
F. The Mini-Computers

1. Introduction

The final topic for discussion is the on-line mini-computers themselves. There are two aspects of this topic to be considered: real-time monitors and programming languages. The fact that the Omega on-line mini-computers have a very small configuration influenced both of these topics to a large degree.

2. The on-line monitor

The Omega PDP-11 on-line monitor is not a true operating system as that term is commonly understood. On such a small computer, it would be impossible to even consider a complete operating system. Rather, the monitor consists of a collection of modules that include I/O drivers for the teletype, paper tape reader and CAMAC interface, commonly used conversion routines, routines for using the Tektronix 611 display (see table 4), an interrupt handler, an on-line controller, and a teletype command interpreter. This interpreter accepts simple commands for loading absolute binary paper tapes, for starting and stopping program execution, for simulating event and burst interrupts, and for interrogating and/or modifying any register or memory cell dynamically. The monitor is simple but provides a basic tool which can be easily used and understood by a wide range of users.

The on-line controller provides a primitive foreground/background type of operation. It requires the user program to be structured into three parts: an initialization sequence (main program), an event procedure, and a general procedure. The initialization sequence is activated once, when the execute command is entered on the teletype, typically at the beginning of a run. This sequence sets up data values, histograms, CAMAC addresses, etc. For the run, informs the on-line controller of the location of the event and general procedures, and then relinquishes control
to the on-line controller. The controller can respond to an event trigger in one of two ways: it can call a system routine to read the counter values into memory using a table of CAMAC addresses set up by the initialization sequence, then write them into the CAMAC buffer between the PDP-11 and the EMR 6130; or it can transfer control directly to the user's event procedure where the data is read under programmer control, and any other type of calculation performed. Typically, this involves the histogramming of experimental parameters such as counter values, hodoscope patterns, etc. The user's general procedure can be activated when a system flag bit is set by the event procedure, at regular intervals by using the monitor clock, or when the operator issues a teletype request to execute it. Status information in a monitor communication cell permits the user to determine what caused his routine to be activated and to take appropriate action. Typically this action will consist of displaying a histogram or set of histograms, typing out parameter values, changing parameter settings, etc. Each user is free to include as many general features in this procedure as he wishes.

3. Programming languages

At the time the idea of purchasing mini-computers for each experiment was being considered, a very important part of this discussion concerned the problem of what language would be used to program these machines. Although the debate centered on the choice between FORTRAN, a "High-Level" language that is widely used on many computers, and the PDP-11 assembly language, a "Low-Level" language, it was resolved by the introduction and acceptance of a third alternative, PL-ll, an "Intermediate-Level" machine-oriented language. This language was designed and implemented by Omega with the intention of developing a better programming tool for PDP-11 computers than either FORTRAN or assembly language. The idea behind PL-ll is simply to keep the high-level
syntax of a FORTRAN-like language, thereby making it easy to learn and program, but to design the language constructs with the architecture and organization of the PDP-11 in mind, thereby making it efficient and close to the PDP-11 hardware.

In order to accomplish this goal, one must consider the advantages and disadvantages of high-level languages (see table 6) and machine-level languages (see table 7). The advantages of high-level languages all derive from the fact that they are "people-structured"; that is, constructs in the language, such as arithmetic expressions, iteration statements, conditional expressions, etc, correspond to the way people think about a particular algorithm. In some sense these represent a natural notation for expressing formal algorithms. Although it can hardly be argued that people think in terms of array declarations and read and write commands, at least these are expressed in a notation that is easy for people to use and understand. The net effect is that programs written easily and reasonably quickly in a high-level language are also easy for both the original programmer and other people to understand and modify.

The problem with high-level languages is that their constructs may bear little if any relationship to the organization of the machine hardware. Too often computer hardware is designed without regard for how it can be used by people, so that the mapping between high-level language constructs and the machine instructions is very bad. The result is that object programs are both large and slow, and for real-time mini-computers, this is unacceptable. Consequently, assembly languages are too often used in order to gain run-time efficiency at the expense of programmer efficiency. A second major reason for the use of assembly languages is that many hardware "features", especially custom-built on-line equipment, may not be accessible in a high-level language.

There is little doubt that an assembly language is an
extremely primitive tool for programming a computer, even a mini-
computer, because it is "machine structured" rather than "people
structured". It requires enormous attention to petty detail that
is irrelevant to the user's task, so that programming becomes a
tedious and error-prone process that dissuades many people from
even attempting it. Thus a great deal of the power of a computer
is not utilized, simply because it is too difficult for people to
write and understand assembly language programs. Large assembly
language programs are virtually unmodifiable, since even the
original programmer will be loath to attempt a change once he
has the whole thing finally in some sort of working order. To
expect a non-professional programmer to use assembly language is
to expect him not to use the computer.

The goal of PL-11 was therefore to be as "high-level" as
possible without sacrificing run-time efficiency. The result is a
language with a syntax based on ALGOL 60 (see the example in
figure 11). It is free-field, with statements and declarations
separated by semi-colons and comments allowed anywhere. The lan-
guage includes declarations, procedures, and control statements,
such as loops, and branching statements, that are very ALGOL-like
in their syntactic form. Arithmetic expressions are more primitive
than ALGOL expressions, because they must be evaluated in a linear
fashion on the PDP-11, but they do retain the well-known arithmetic
operator notation (+ for add, - for subtract, etc.). All register
and storage allocation is under programmer control, and subscripting
is restricted to constants or constants plus a register, simply
because this is the only type of indexing which can be handled
directly by the hardware. All eight of the PDP-11 addressing modes,
including indirect addressing and stacking operations, are directly
available to the programmer. A heavily used feature of PL-11 is
the ability to give meaningful symbolic names of any length to
almost any object of interest in the computer: fixed memory cells,
compile time constants, elements of an array, registers, I/O
devices, CAMAC registers, etc. This plus the ALGOL-like structure of the language elements encourages a very readable programming style with programs that can be easily understood and modified. In addition, many of the more common assembly language errors are avoided completely due to the discipline imposed by the language structure. This means that programmers can get PL-11 programs running and debugged much faster than equivalent assembly language codes.

Furthermore, PL-11 is efficient (see an example of the code generated in figure 12). All instructions in the computer can be represented by language constructs, so that all hardware features are available to the programmer. More importantly, the code generated by the compiler for each language construct is part of the specification of the language itself, thereby enabling the programmer to be as conscious as he wishes of the number of instructions he generates. However, experience has shown that a readable programming style with good global organization that can be easily understood and modified results in far more efficient programs than local nit-picking tricks can ever hope to attain.

In conclusion, the Omega experience indicates that the best tool for programming a mini-computer is neither FORTRAN nor assembly language, but a machine-oriented intermediate language. This tool can be used by systems programmers and applications programmers alike. PL-11 has been used extensively by all the on-line physics groups with the result that the PDP-11 is programmed easily and efficiently by physicists for a wide range of tasks, many of which might not have been attempted with either FORTRAN or assembly language.
Table 1: Criteria for selecting a computer

<table>
<thead>
<tr>
<th>Category</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benchmark tests</td>
<td>compiler - size, speed, reliability</td>
</tr>
<tr>
<td></td>
<td>object code - size, speed</td>
</tr>
<tr>
<td></td>
<td>operating system - usability</td>
</tr>
<tr>
<td>Meeting the specifications</td>
<td>expandability</td>
</tr>
<tr>
<td></td>
<td>availability of components</td>
</tr>
<tr>
<td>Reliability</td>
<td>hardware</td>
</tr>
<tr>
<td></td>
<td>software</td>
</tr>
<tr>
<td></td>
<td>documentation</td>
</tr>
<tr>
<td>Maintenance</td>
<td>availability</td>
</tr>
<tr>
<td></td>
<td>quality</td>
</tr>
<tr>
<td></td>
<td>hardware, software, peripherals</td>
</tr>
<tr>
<td>Documentation</td>
<td>existence</td>
</tr>
<tr>
<td></td>
<td>accuracy</td>
</tr>
<tr>
<td></td>
<td>availability of source code for system software</td>
</tr>
<tr>
<td>Price</td>
<td></td>
</tr>
</tbody>
</table>
Table 2: Functions that an on-line physics data handling system must perform simultaneously

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Development of data handling system</td>
</tr>
<tr>
<td></td>
<td>(online hardware and software, offline software)</td>
</tr>
<tr>
<td>2</td>
<td>Development of on-line physics systems</td>
</tr>
<tr>
<td></td>
<td>(hardware and software)</td>
</tr>
<tr>
<td>3</td>
<td>Real-time data acquisition, control, and sample analysis</td>
</tr>
<tr>
<td>4</td>
<td>Data reduction (pattern recognition)</td>
</tr>
<tr>
<td>5</td>
<td>Data analysis (geometry and kinematics)</td>
</tr>
</tbody>
</table>
Table 3: The data-link routines

CII (FORTRAN USER LEVEL)

CNILTK (LINKNO, PROGRAMID, NBYTES, KIND)
DCNTLK (LINKNO)
WRITLK (LINKNO, BUFFER, NBYTES, STATUS, IDOPRECEIVER)
READLK (LINKNO, BUFFER, NBYTESEXPECTED, NBYTESRECEIVED, STATUS)
RDMLUK (LINKNO, BUFFER, NBYTESEXPECTED, NBYTESRECEIVED, STATUS)
idlelk

EMR (FORTRAN USER LEVEL)

CNCTLK (EMRPROGRAMID, CIIPROGRAMID)
DCNTLK (EMRPROGRAMID)
WRITLK (BUFFER, NBYTESTOSEND, STATUS)
READLK (BUFFER, NBYTESRECEIVED, STATUS)

PDP

CNCTLK (PDPPROGRAMID)
DCNTLK (PDPPROGRAMID)
WRITLK (BUFFER, NBYTESTOSEND, STATUS, IDOPRECEIVER)
READLK (BUFFER, NBYTESRECEIVED, STATUS, IDOFSENDER)
Table 4: PDP-11 on-line monitor routines for using the Tektronix 611 storage tube display

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DYERAS</td>
<td>&lt;erase the storage screen&gt;</td>
</tr>
<tr>
<td>DYPONI</td>
<td>&lt;plot a table of N points&gt;</td>
</tr>
<tr>
<td>DYVECT</td>
<td>&lt;plot N vectors from a table of N+1 endpoints&gt;</td>
</tr>
<tr>
<td>DYCCHAR</td>
<td>&lt;plot a string of N ASCII characters&gt;</td>
</tr>
<tr>
<td>DSREGN</td>
<td>&lt;return coordinates of specified region of screen&gt;</td>
</tr>
</tbody>
</table>

Table 5: 1-Dimensional histogram routines on all Omega computers

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HBOOK1</td>
<td>(HISTOGRAMNUMBER, NWORDSINTITLE, TITLE, NBINS, LOWEREDGE, BINWIDTH, STATUS)</td>
</tr>
<tr>
<td></td>
<td>&lt;reserve space for 1-D histogram&gt;</td>
</tr>
<tr>
<td>HFIILL1</td>
<td>(HISTOGRAMNUMBER, VALUE)</td>
</tr>
<tr>
<td></td>
<td>&lt;increment bin in 1-D histogram&gt;</td>
</tr>
<tr>
<td>HSTDOL</td>
<td>(HISTOGRAMNUMBER, OUTPUTUNIT, POSITION)</td>
</tr>
<tr>
<td></td>
<td>&lt;display 1-D histogram on output unit&gt;</td>
</tr>
<tr>
<td>HZERO</td>
<td>(HISTOGRAMNUMBER)</td>
</tr>
<tr>
<td></td>
<td>&lt;reset all bins of this histogram to zero&gt;</td>
</tr>
<tr>
<td>HFREE</td>
<td>(HISTOGRAMNUMBER)</td>
</tr>
<tr>
<td></td>
<td>&lt;release space occupied by this histogram&gt;</td>
</tr>
<tr>
<td>HMENU</td>
<td>(OUTPUTUNIT)</td>
</tr>
<tr>
<td></td>
<td>&lt;display list of all current histograms&gt;</td>
</tr>
</tbody>
</table>
Table 6: Advantages and disadvantages of high level languages

<table>
<thead>
<tr>
<th>Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Easy to learn, use, read, modify</td>
</tr>
<tr>
<td>2. Self-documenting</td>
</tr>
<tr>
<td>3. Transportable</td>
</tr>
<tr>
<td>4. Fast to program and debug</td>
</tr>
<tr>
<td>5. Usable without knowledge of the machine</td>
</tr>
<tr>
<td>6. Less error prone</td>
</tr>
<tr>
<td>7. Designed to encourage good program structure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Inapplicable</td>
</tr>
<tr>
<td>2. Cannot utilize hardware features</td>
</tr>
<tr>
<td>3. Inefficient object code</td>
</tr>
<tr>
<td>4. Not really transportable</td>
</tr>
</tbody>
</table>
Table 7: Disadvantages and advantages of machine level languages

<table>
<thead>
<tr>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Require a detailed knowledge of the machine</td>
</tr>
<tr>
<td>2. Prone to &quot;tricks&quot;</td>
</tr>
<tr>
<td>3. Difficult to learn, use, read, modify</td>
</tr>
<tr>
<td>4. Not self-documenting</td>
</tr>
<tr>
<td>5. Not transportable</td>
</tr>
<tr>
<td>6. Tedious to program and debug</td>
</tr>
<tr>
<td>7. Highly error prone</td>
</tr>
<tr>
<td>8. Completely unstructured</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Code produced is one-for-one</td>
</tr>
<tr>
<td>2. All hardware features are usable</td>
</tr>
<tr>
<td>3. Efficiency is a function of the programmer.</td>
</tr>
</tbody>
</table>
The initial system

Fig. 1
Fig. 3  The CII 10070 configuration
Fig. 4  The EMR 6130 configuration
Fig. 5  The 3 system
Fig. 6  A possible expansion of the 3 system
Fig. 7  A second possible expansion of the 3 system
Fig. 8 The 4 system
<table>
<thead>
<tr>
<th>Phase</th>
<th>Initiating Computer</th>
<th>Responding Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase 1</td>
<td>Initiate</td>
<td>Respond</td>
</tr>
<tr>
<td>Phase 2</td>
<td>Write</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td>End of Block</td>
<td>Accept / Reject</td>
</tr>
<tr>
<td>Phase 3</td>
<td>Write</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td>End of Block</td>
<td>Good / Bad</td>
</tr>
</tbody>
</table>

Fig. 10  The data link dialogue
PROCEDURE MAGICSQUARE:
BEGIN
COMMENT ALGORITHM 118 Comm. ACM 5 (Aug, 1962) - Magic Squares:
INTEGER NSQR, N SYN R0, I SYN R1, J SYN R2, IJ SYN R3, K SYN R4

N => J => I + 1 SHA;
FOR K FROM 1 STEP 1 UPTO N => NSQR * N DO
BEGIN
  I => IJ SHA 4 + J SLA;
  IF X(IJ) /= 0 THEN
  BEGIN
    IF I+1 < 1 THEN I + N;
    IF J+2 < 1 THEN J + N;
    I => IJ SHA 4 + J SLA;
  END;
  K => X(IJ);
  IF I+1 > N THEN I = N;
  IF J+1 > N THEN J = N;
END;
END;

Fig. 11 A sample PL-11 program
This program represents the PL-11 code for the example found on page 26 of the DEC PDP-11 Handbook. As written, it will generate exactly the same number of instruction words as that example. The program generates a histogram of the frequency of occurrence of all values in the range 1-100 in 'itable'.

ARRAY 100 INTEGER 8TABLE;
ARRAY 1000 INTEGER ITABLE;

COMMENT NEXT CLEAR THE OUTPUT TABLE TO 0 - NOTE THAT IN PL-11, THIS TABLE WOULD BE AUTOMATICALLY INITIALIZED TO 0 WHEN THE PROGRAM IS LOADED. HENCE, THIS CLEARING IS NECESSARY ONLY IF THE TABLE HAS BEEN USED PREVIOUSLY;

IF 8TABLE => RO; 
FOR R1 FROM +100 STEP 1 UPTO 0 DO 0 => 8POP(R0));

COMMENT HISTOGRAM ALL THE ITEMS IN ITABLE WITH VALUE 0 < VALUE <= 100;

IF 8TABLE => RO; 100 => R2; COMMENT PUT LIMIT IN R2 FOR SPEED;

FOR R1 FROM +1000 STEP 1 UPTO 0 DO

IF 8POP(R0) => R4 > 0 & R4 <= R2 THEN

COMMENT VALUE WITHIN RANGE, ADD IT TO THE HISTOGRAM BIN;

BEGIN R4 SLA; 8TABLE(R4) + 1;

END;

IEND.

Fig. 12 A PL-11 program with generated code
ON BOARD COMPUTERS FOR SATELLITES

M.A. Perry
European Space Research and Technology Centre,
Noordwijk, Netherlands

SUMMARY

The paper is concerned with stored program general purpose digital computers for use on satellites. Environmental and other constraints which render satellite computers different from equivalent ground machines are discussed and several approaches to the realization of such computers are briefly illustrated where three representative architectures are described. Finally the architecture of an on board computer under development by ESRO is described.

1. INTRODUCTION

Before discussing the different approaches which have been adopted to realize On Board Computers it is as well to summarize the environmental and other constraints which determine the properties of the machine. Thereafter we can briefly survey some typical applications for computers on board satellites and describe some machines which have been developed. Finally the structure of a computer being developed by ESRO under contract will be outlined.

In using the term 'computer' we mean more formally a stored program general purpose digital computer where at least some part of the contents of the main memory is electronically alterable. This distinction is necessary since spacecraft sometimes employ quite sophisticated special purpose digital (and analog) systems which may be programmable within certain limits but which lack the full flexibility of a digital computer.

2. BASIC CONSTRAINTS

One of the basic constraints which governs most space activity is the size of the available budget, since space projects are expensive. Typical costs incurred for the launcher and launch support needed to put a payload of around 600 Kg. into a medium height circular orbit are around £4 million.

The consequences of this are that the mass of a satellite must be carefully contained within certain maximum bounds. Furthermore the risk of requiring a second launch because of a failure on board the satellite
must be minimised.

These global constraints reflect on the basic parameters of an on
board computer in the following way.
a) The intrinsic mass of the computer must be low, for reasons already
described.
b) The power consumption must be minimized, since the masses of the solar
array and batteries are roughly proportional to the electrical energy
to be generated or stored.
c) The volume of the computer should be reasonable for its mass; a
density of around 1.0 is typical. A voluminous spacecraft would never
fit under the nose cone of the launcher.
d) Finally the reliability of the computer must be exceptionally high,
and in this respect Mean Times Before Failure (MTBF) of between 15 and
75 years are talked of.

Table 1 shows some typical mass budgets for scientific satellites which
have been studied in ESRO at various times. Note that a typical on board
electronics package occupies around 17% of the total mass, and itself
weighs a few tens of kilograms.

<table>
<thead>
<tr>
<th></th>
<th>GEOS</th>
<th>HELOS</th>
<th>SORFL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probable launch</td>
<td>1975</td>
<td>1978</td>
<td>1980's</td>
</tr>
<tr>
<td>Electronics (excl. expt)</td>
<td>25.3</td>
<td>39.2</td>
<td>46.2</td>
</tr>
<tr>
<td>Solar Array</td>
<td>10.5</td>
<td>6.2</td>
<td>32.0</td>
</tr>
<tr>
<td>Remainder</td>
<td>114.2</td>
<td>151.6</td>
<td>206.8</td>
</tr>
<tr>
<td>Total</td>
<td>150</td>
<td>197.0</td>
<td>285.0</td>
</tr>
<tr>
<td>Electronics % of Total</td>
<td>17</td>
<td>20</td>
<td>16</td>
</tr>
</tbody>
</table>

The power budgets given in Table 2 show the distribution of electrical
power amongst the various sub-systems. The exact distribution changes,
but we should note that total power budget for a spacecraft is consider-
ably less than the power consumed by many mini-computers.
Table 2
Power Budgets

<table>
<thead>
<tr>
<th></th>
<th>GEOS</th>
<th>HELOS</th>
<th>SOREL</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Telemetry, Tracking and</td>
<td>20.9</td>
<td>53.0</td>
<td>96.0</td>
<td>W</td>
</tr>
<tr>
<td>Command</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attitude Control System</td>
<td>24.5</td>
<td>26.0</td>
<td>4.8</td>
<td>W</td>
</tr>
<tr>
<td>Experiments &amp; Miscellaneous</td>
<td>36.8</td>
<td>8.5</td>
<td>46.2</td>
<td>W</td>
</tr>
<tr>
<td>Total</td>
<td>82.2</td>
<td>87.5</td>
<td>146.0</td>
<td>W</td>
</tr>
</tbody>
</table>

From these figures we can draw some rough conclusions about the global characteristics of a suitable satellite computer.

- low mass: 1 to 10 kg
- low power: 1 to 20 W
- low volume: 1 to 10 L
- high MTBF: 15 to 75 years

3. THE SPACE ENVIRONMENT

The space environment plays an important part in determining the characteristics of a satellite computer.

The first demand on the endurance of space borne equipment occurs during the launch when it is subjected not only to quite strong acceleration but also to a considerable amount of vibration which is generated by the launcher.

Although the launch only requires perhaps 30 minutes, equipment is subjected to a great deal of stress during this time and must be designed to withstand this.

Space itself is notable for being more or less a vacuum. Additionally, solar energy will be incident on the spacecraft. Whilst some of this energy is converted into electrical energy by the solar array, the rest will be absorbed by the spacecraft itself. Convection cooling is not possible and a combination of conduction and radiation cooling must be used to cool the spacecraft. If the spacecraft goes in and out of eclipse frequently it will be subject to thermal cycling. For these reasons equipment having a low power consumption is favoured since it reduces the likelihood of 'hot spots' which cause premature failures, and designs must be able to withstand the effects of thermal cycling.
Although space may be regarded as a vacuum, it nevertheless contains considerable quantities of energetic charged particles, some having energies of a few MeV\(^1\). Sources of charged particles are various, and their origin is beyond the scope of this paper, but we can note some of the more important ones, such as the van Allen Belts and the Solar Wind. Proton densities can be as high as \(10^8\) per cm\(^2\). Such levels of radiation can have a damaging effect on semiconductor devices and degrade their performance under certain circumstances.

The above factors make their presence felt in the choice of technologies which are used for the manufacture of spaceborne equipment. These technological constraints weight heavily in determining the characteristics of a satellite computer, in particular its computational power. We must be careful not to regard the problems of technology as trivial for experience has shown that the technological problems encountered in the conception and design of spaceborne equipment are at least as difficult as those of systems design, and require a considerable insight into the physics of materials.

4. RELIABILITY

Good reliability can be achieved by a two pronged approach. The first of these prongs involves careful manufacture of all parts, supported by good inspection and quality control. This activity requires not only visual inspection but also the use of highly sophisticated equipment such as Scanning Electron Microscopes or X-Ray Monitors. A good understanding of the physics of failure is paramount if the best technologies are to be chosen and good results obtained. Most high reliability components carry with them a good deal of documentation which outlines the life history of that component and the tests which it has undergone. In this way defective batches can be easily traced at any time.

The second prong of the approach is in the design of the system and the hardware itself. By first keeping the system as simple as possible a good deal of potential problems are avoided. System reliability can be further enhanced by introducing additional redundant circuitry which will take over the functions of the circuit which has failed.
Redundancy may be in two forms:

1) Masking Redundancy (fig. 1a).
   In this case three identical units a, b and c operate continuously in parallel. A voting circuit computes the majority vote (2 out of 3) and ignores the output of the dissenting unit.

2) Stand by Redundancy (fig. 1b)
   Three units a, b and c operate one at a time depending on which has been chosen by the switches. The other two stand by but unpowered.

To summarize the trade-offs between the two techniques stand by redundancy offers longer life, but service will be interrupted in case of failure. Masking redundancy offers continued service but over a shorter overall lifetime.

5. OPERATIONAL REQUIREMENTS

The functions performed by a satellite computer may be likened to a certain extent to those performed by a small process control computer. Fig. 2 outlines a rather generalised statement of typical applications.

The computer is connected to various satellite subsystems or experiments which provide raw or pre-processed data. Some of these systems may be controlled by the computer in a closed loop system, whereas others simply produce data which should be processed prior to being sent to ground.

Communication between ground and the computer will be over the normal satellite telecommand and telemetry links. The capacity of these links is weak compared to what is possible on the ground mainly due to limitations which must be imposed on the design of the on board antenna and transmitter. Typical bit rates for the up link are in the range 90 to 1500 bits per second, but because of the use of error detecting codes and synchronisation words the rate of basic information transfer is considerably less, of the order one quarter. A wider range of possibilities exists for the down link where bit rates may lie between 12 bits per second to tens of kilobits or even more depending upon the characteristics of the on-board communications package and the mission. Format efficiencies of 90% are typical and a great deal of the capacity of a down link can be used to carry information.

The telecommand up link can be used for sending on/off commands to
the computer and for reprogramming the memory during flight. This is a highly desirable feature to have, since it greatly adds to the flexibility of the machine, but it introduces problems of ensuring that memory data is correctly received and loaded in the correct address. Programs should occupy as little memory space as possible since this not only reduces the weight of the flight memory, but also reduces the time needed to load a program, which could be as long as 20 minutes per thousand words in the worst case. For this reason high level languages such as FORTRAN or ALGOL are not preferred and programs are coded in mnemonic code and assembled on the ground prior to transmission to the satellite.

If the computer should fail, we must be able to diagnose the fault from the ground by way of the existing radio links. Simple instruction test routines are not always enough. The diagnostic information must be sufficient to pinpoint the fault with some degree of accuracy in case in-flight reconfiguration of the computer might restore it to operation in a degraded mode. It is also useful to perform in flight diagnostics in a routine manner at regular intervals and telemeter the results to ground to establish confidence in the computer.

6. APPLICATIONS

The possible applications for a general purpose computer on a spacecraft are many and varied. These tasks include:

- Attitude and Orbit Control
- Logging of routine 'housekeeping' data
- Compression of Data
- Formatting of Data for Telemetry
- Control of Experiments
- Experiment Data Handling
- Spin Synchronous demodulation
- Fast Fourier Transform
- Pulse Height Analysis
- Computation of averages etc. etc.

7. SOME TYPICAL ARCHITECTURES

The first general purpose digital computer to be flown in a manned spacecraft was the Gemini-computer\(^2\). Word length was 26 bits and the Add time 140 s. At a later date, the Apollo Guidance Computer\(^3\) was flown. This machine was faster, having an Add time 23.4 us. Both machines used considerable amounts of read-only memory for program storage to prevent
loss of program during flights although read/write storage was used for data. Thus, it was only possible to alter the program in flight limited way.

7.1 SDP-3 Computer

Probably the first stored program general purpose digital computer to be flown of a Scientific Satellite was the SDP-3 computer which was put on Imp-1 launched in March 1971. The machine is a 16 bit, serial, single word instruction, single address machine. The memory is divided into pages of 256 words, which can be directly addressed, and one level of indirect addressing is available. One index register is provided.

Since this computer was ultimately intended as a common user system, there exist two modes of operation, namely 'user mode' and 'monitor mode'. To prevent one user program from inadvertently interfering with another, interpage access is prohibited in the user mode.

The computer contains a set of 54 instructions covering all the normal operations usually expected in a small computer. Arithmetic is done in two's complement, fixed point format, but multiplication, division and floating point or double precision operations must be executed through software. Input/output operations are carried out via memory rather than an accumulator with several possible instructions.

XMIN Exchange memory with output no execute pulses.

These instructions enable data transfer in either direction with experiments. In order to furnish the address of the experiment, a control register which controls a multiplexer must be loaded.

LDCA Load control register A

There also exists a LDCB instruction. The difference between the A and B control registers is that the contents of B are cancelled after the end of the instruction which sets it, whereas the contents of A must be changed by a second load instruction.

16 levels of interrupt are provided. Besides performing the normal forced jump, interrupt also enables power to be saved. After the execution of a HALT instruction, all power to the computer is turned off with the exception of the clock and priority interrupt circuitry.
An output channel remains powered if it is enabled, and the power to the input channel is controlled by the PCM Telecommand System. If either channel is needed for data transfer, power is turned on for one instruction period only. Power is restored to the computer on receipt of a priority interrupt request.

All instructions require equal times for execution, namely 78 microseconds. Whilst low speed is inherent due to the choice of a serial data flow, it also offers certain advantages.

Since fewer memory accesses are made per second, there is a corresponding reduction in memory power consumption. Furthermore, serial data flow requires less circuitry to switch data than would be needed with a parallel organisation, and this leads to savings in mass and power consumption.

The SDP-3 is built with low power DTL circuits and consumes around 9.0 Watts when operating. The overall mass including a 4096 word core memory is around 3.6 Kg. A system block diagram is shown in fig.3.

7.2 A Computer Employing Redundancy

The SDP-3 and machines such as the On Board Processor designed for the OAO satellite did not contain large amounts of redundancy. As a consequence their estimated lifetimes are comparatively short, although sufficient for a mission lasting between one and two years. When a long mission - such as a grand tour - is considered a new approach must be made in order to achieve an acceptable reliability. One typical system which 'employs stand-by redundancy' is the Multipac Computer (fig.4), which can be reconfigured by telecommand if a particular unit should fail.

Up to 15 Semiconductor memories of 4096 words of 12 bits are provided. The storage medium is Complimentary MOS and is therefore volatile i.e. stored data will be destroyed if the power is lost. Since standby power is of the order of $10^{-7}$ watts per bit implying a total memory standby power of around 30 milliwatts, the problem of volatility can be solved with a battery.

Up to 5 identical processor units are provided, but full multiprocessing capability does not exist since no hardware exists to resolve conflicts when two processor units try to access the same memory. Nevertheless, communication between processors is effected in a simple manner. When large amounts of data are transferred, this is done by
switching memories after communication between processors. For small amounts of data, a transfer via I/O - G.P. registers is easier since all processors can address all registers. If two processors attempt to write into the same register, the logical OR of the two inputs is written. Simultaneous reading of the register is possible however.

The system can handle a total of 63 Input/Output registers if needed but not all this capacity is necessary for normal operations.

The instructions may be of one or two words in length. The first word contains a 6 bit operation code and 6 bit index register address. The second word contains the memory address, if a memory reference instruction is called. Instruction time is of the order of 15 µs, and serial data transfer is used.

Multipac will be built using large scale integrated circuits throughout and estimates of system lifetime depend on the reliability of these circuits. Assuming a failure rate of $10^{-6}$ per hour per circuit, the probability of the minimal system surviving 3 years is 92.5%. The estimated power consumption for a system having 6 memory blocks and 3 processors is about 16 watts and the fully expanded system requires around 32 watts.

7.3 **STAR Computer**

'STAR'\(^8\) means 'self test and repair'.

This highly advanced machines has been under development in the USA for some years now and is planned for deep space missions where flight times will be several years. Because of the great distances which such spacecraft travel from the earth, not only is the propagation delay for electromagnetic waves quite significant, but also the available bandwidth of spacecraft/ground radio links very narrow. Consequently a telecommand reconfigurable system will not suffice, and a self test and repair design philosophy becomes necessary.

The overall system is depicted in Fig.5. Two bus bars are used for communication.

- memory input bus
- a memory output bus

Connected to these buses in standby redundancy are a number of operational units.
COP  Control Processor which contains the location counter and index registers and carries out address modification prior to execution.

LOP  Logic Processor which performs logical operations on data words.

MAP  Main Arithmetic Processor

ROM  16 383 words of Read Only Memory

RAM  Block of 4096 words of Read Write Memory — up to 12 blocks are directly addressable

IOP  Input/output Processor

IRP  Interrupt Request Processor

TARP  Test and Repair Processor which monitors the operation of the computer and implements recovery from a failure.

The functions of the Test and Repair Processor are two fold. Firstly it checks the operation of the computer instruction by instruction. Data sent along the memory data buses are encoded with error detecting codes and the TARP checks the validity of this data. Status messages from functional units are checked for predicted responses. An incorrect word or a deviation from a predicted response causes an interruption of normal computing and an entry into the recovery mode of operation.

The TARP contains an address register, which holds a 'roll back address' which is the point in the program at which normal operation is resumed upon recovery. This register is updated under program control. Prior to updating, the contents of all processor registers needed for recovery are stored in duplexed memory units. On receipt of an error message the program is rerun from the roll back address, but if a repeated fault indication is received from the same unit, it is replaced by switching off the faulty unit and switching on a new one in parallel.

Since the TARP is the heart of the system it is operated in triplicate (with some standby spares) and a majority vote is taken. If a dissenting TARP is discovered, recovery mode is entered. If the dissenting TARP continues to disagree with the majority, it is automatically replaced by a spare unit.

The STAR Computer is 4 bit byte serial machine which uses a 32 bit instruction word, which due to the use of error detection coding offers a 16 bit address and 216 valid Op-Code variants. The instruction set uses 130 opcodes and includes 180 single address instructions about one
third of which are indexable. The basic machine cycle occupies 5\(\mu\)s and instructions require two or more cycles for execution.

**8. ESRO ACTIVITIES**

Following some initial activity within ESRO, a contract was placed with Selenia Industrie Elettroniche Associate SpA of Rome, Italy in late 1970 for the study and simulation of a small, flexible general purpose, stored program digital computer for use on board orbiting spacecraft\(^9\). The study called for the definition of a machine with the following parameters:

- **Weight**: 5 Kg
- **Power Consumption**: 10 W
- **Reliability**: 90% over 18 months

A simulator was requested, since it provides a valuable tool for the study and development of programs without recourse to a prototype on board computer. The ease with which a simulator can be reproduced (it is only a deck of cards) allows it to be made available in many places at low cost. An assembler was requested as a standard item of support software.

In its finally defined form the computer has evolved as a system composed of several modules, interconnected via a data transfer controller. This concept enables the same hardware which is used for a simple computer for a small spacecraft, to be expanded into versatile and powerful multiprocessor for a large space vehicle with little extra design effort. In addition, the use of microprogramming permits the instruction set to be tailored to the requirements of a mission with only minor changes in hardware. These two concepts lay the foundation of the design philosophy developed during the study.

Work commenced with definition of 'Elementary Functions' performed by the computer. These were not instructions, but rather complete operations such as 'save register content'. The various tasks were analyzed in terms of elementary functions and frequency tables were synthesized to show the relative rate of usage.

In conjunction with ESRO, Selenia defined three model missions, representing a best estimate of requirements of future scientific and application satellites at that time, (February 1971). Each mission
consisted of a group of tasks, and the computational requirements for these missions were estimated. It was now possible to evaluate the efficacy of candidate instruction sets in the light the task analysis, and a preferred set was chosen, subject to review during the definition of system and hardware. At this stage, the following basic features of the architecture became clear.
- The use of a 16 bit word length for data and instructions was viable.
- The machine should be parallel in order to achieve the necessary computational speed.
- The add time should be in the range 2.5 to 7 microseconds
- Direct Memory Access channels were needed.

Following an investigation of several candidate architectures it was concluded that the computer should have the following features:
- Microprogrammed control unit for flexibility
- Two's complement arithmetic
- 512 addressable Input/Output channels
- Interrupts expandable in blocks of 8
- Single address, 16 bit instructions
- 1.4 microsecond cycle time
- Memory expandable to 64K words

In addition, the system features a transfer controller to superintend the transfer of data between memory and other devices such as Central Processor Units (CPU) and Direct Memory Access Units (DMA) (Fig.6). Each unit is allocated a priority with which the controller shall treat an application for a data transfer between memory and the unit, and units are serviced in order of priority. The main advantage of this approach is that the computer system may be constructed from as many CPU's, DMA's Memories and other units as may be needed, requiring no more than minor modifications to the transfer controller, which is a fairly simple item anyway. In its simplest form the computer would consist of the following: Transfer Controller
CPU
DMA for telecommand
Memory
DMA for telemetry
DMA for experiments (if needed)
In addition spare CPU's and memory blocks can be added as redundant spares or to increase the computational power of the system (fig.7) The CPU consists of three main blocks: (Fig.8)
- Operating Unit
- Control Unit
- Program interrupt unit

The operating unit performs the arithmetic and logical operations and stores the results in working registers. It has for this purpose an upper (A Register) and lower accumulator (B Register), program counter (PC), memory address register (AR), one index register (X) and a 'data register' (DR) for storing intermediate results. Trade-off studies showed that little benefit accrued from having more than one index register. The ALU is a combinational logic which operates on two data streams to perform ADD, SUBTRACT, AND, Exclusive-OR and other logical operations.

Status registers - for example - overflow-, carry- and interrupt mask registers are contained in the control unit. This unit uses micro-programming to generate control signals which time and regulate the flow of data within the computer. The instruction set is stored as a micro-program, which is contained in a small semi-conductor read-only memory. By changing this read-only memory, we change the instruction set.

The on-board computer has 61 instructions in the set and includes the capability indexing and indirect addressing. A 9 bit address field is used in a fixed page organisation for memory reference instructions.

A trade off study of available integrated circuit logic families showed low power TTL to be the preferred technology with many advantages, not the least being that these circuits are available already, and expensive device development work is avoided.

It was further shown that, in order to meet the power consumption constraint (10 Watts) a plated wire memory having a pulsed power supply was needed. An analysis of the failure rates of each portion of the computer predicted the memory to be the least reliable functional block and the supply of a reliable plated wire memory is therefore an important point. Failure rates were computed using data taken from European and U.S. sources and the Contractor's past experience.
The reliability predictions for an 18 month period are given below.

<table>
<thead>
<tr>
<th>Component</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA units</td>
<td>better than 0.99</td>
</tr>
<tr>
<td>CPU</td>
<td>0.97</td>
</tr>
<tr>
<td>Memory</td>
<td>0.88</td>
</tr>
</tbody>
</table>

Computer complete with one stand by Memory 0.94

Power consumption is estimated at 10.6 Watts for the complete computer which compares favourably with the target of 10.0 Watts. The mass of the simplest system is predicted at 4.5 Kg and this increases to 6.5 Kg if a redundant memory is added.

9. CONCLUDING REMARKS

In the small amount of space available, it has not been possible to give an exhaustive treatment of all the aerospace digital computers which have been studied, designed and built.

Rather we have highlighted those which have been thought to be typical of their class. Those who wish to pursue the subject further are referred to work cited in references 10 to 14 which includes efforts both in Europe and the USA.

REFERENCES

1) Earths Particles and Fields, Editor Billy M. MacCormac, Published Reinhold.

2) 'Gemini Electronic firsts', Electronics, May 3 1965, page 71-76


4) Rodger A. Cliff, 'The SDP-3 a computer designed for Data Systems of Small Scientific Spacecraft'. NASA X-711-69-43

5) Rodger A. Cliff, 'Recent Improvements in the SDP-3' NASA X-711-69-320


12) H. Felix, Projet de Calculateur Embarqué sur le Satellite Roseau, CNES International Conference on OnBoard Computers on Rockets and Satellites, Paris December, 3-6 1968

13) Ch. Habenicht, L. Hohnberg, R. Klett, B. Maag et al. 'Systemkonzept eines Bordrechners für Raumflugkörper BMBW - FBW7011

Fig. 1a  Masking redundancy

Fig. 1b  Stand-by redundancy
Fig. 2 On board computer ground satellite communication
Fig. 4  Multipac block diagram
Fig. 5 Star computer organisation
Fig. 6  On board computer system block diagram
Fig. 7 Simple redundant or multiprocessor configuration
BIRTH AND LIFE OF THE BOL-SYSTEM

J.E.J. Oberski

Institute for Nuclear Physics Research
Amsterdam, the Netherlands

INTRODUCTION

This paper deals with the BOL-system as a case study of a computer application in nuclear spectroscopy. "Spectroscopy" however, in a somewhat broader sense than spectroscopists would mean. The outline of the paper is:

1) What is IKO
2) What is the BOL-system
3) What kind of experiments are performed with BOL
4) What does the computer configuration look like
5) How is it used in experiments
6) How did it grow historically
7) Some ideas of builders and users now
8) Some conclusions
9) Some examples

1. WHAT IS IKO?

At the Institute for nuclear physics research (Amsterdam) are working about 250 people in four research groups: beta-gamma spectroscopy, nuclear reactions, electron scattering and radiochemistry; and eight service groups: mechanical, electronic, digital, software, synchro-cyclotron, electron-linac, domestic and administrative. The two accelerators of the Institute are a synchro-cyclotron for protons (45-55 MeV), deuterons (23-28 MeV), $^3$He-particles (about 65 MeV) and Alfa-particles (45-55 MeV), and an electron linac (85 MeV). A proposal for a 10% duty factor electron linac of 300 MeV is

$x$) Dr. L.A.Ch. Koerts is gratefully mentioned as the auctor intellectualis and stimulator of the BOL-system, out of the great number of people who were and are involved during its building and usage.
still waiting for governmental approval. Concerning computers: each research group by now has got its own DEC-PDP8 of one kind or another while there is a spare one, mainly in use for off-line development of interfaces. Besides, the Institute has got a medium size computer, a Philips-Electrologica X8, which is an essential part of BOL.

2. **WHAT IS THE BOL-SYSTEM?**

BOL, the dutch word for "sphere", is used to measure coincidences between charged particles emerging from nuclear reactions in a target during bombardment with one of the extracted beams of the synchro-cyclotron. The maximum solid angle around the target, with detection sensitive surface is 10\(^\circ\), meaning 1% mean coincidences efficiency, but these figures of course are depending on the specific experiment. Parameters that are registered for every particle of a coincident event are: particle type, energy and angle. Since all different kind of particles, at different energies and angles are measured simultaneously, processes are automatically normalized relative to each other. Events are stored on magnetic tape for off line analysis afterwards (fig. 1, 2).

The following sections will deal with:

1) the scattering chamber
2) the detectors
3) the measuring electronics
4) the computers

2.1 **The scattering chamber**

The scattering chamber consists of several layers. The inner one is a spherical shell of 16 cm diameter, containing the target in its center. It has some 64 holes into which detectors with some electronic circuitry are inserted. The shell is cooled down during operation to -20\(^\circ\)C. The second layer is a vacuum cooling shell of about 70 cm diameter. More electronics is directly connected to the inside-vacuum part here. The last layer of about 120 cm diameter serves
as a reservoir for aircooling of the outside-vacuum electronics. The holes in this cooling sphere were machined to 50 µm insuring accurate positioning of the detectors.

2.2 Detectors

Two sorts of charged-particle detectors are used in each of the 64 holes: a stack of about 4 mm thick Si(Li) PIN ones and a position sensitive gold layer surface barrier detector the so called checkerboard detector of .3 mm thickness. The latter contains perpendicular strips on front and backside, providing a 10 angular resolution. The checkerboard detector is used at the same time to determine the angle of emission of a particle and its energy loss. The total stack measures the total energy of incoming particles, since they are stopped. These two energy values serve also to identify the particle type.

2.3 Measuring electronics

We distinguish three main units in each detection-channel: detection unit, ADC-unit and logic unit. The detection unit contains the charge sensitive amplifiers still inside the vacuum of the scattering chamber and some more amplifiers and shapers outside the vacuum, a.o. for generation of position indication signals. Logic signals are sent to a logic unit which will help to determine coincidence relations and eventually store the data. Analog signals are sent to an analog unit which eventually will digitize them and store them also. One particle then is characterized by a word of 72 bits. The logic and ADC units are housed in "tuns", the latter containing also a coincidence unit and part of the computer interface unit, the sequencer.

2.4 Computers

A PDP8 is connected to the experiment. It takes data and stores them on magtape or it sends data directly to the EL-X8 computer, a facility which is essential in the stage of setting up an experiment. A second PDP8 can be used for data
taking instead of the first one. It has no magtapes, but can also direct the data to the X8. Data are displayed by means of a storage scope or line printer.

3. WHAT KIND OF EXPERIMENTS ARE PERFORMED WITH BOL

The main feature of the experiments is, that they are kinematically complete, meaning that at least as many parameters are measured, as is the number of degrees of freedom for a particular experiment. Also data do not cover only the reaction plane, but many different angles. Some topics are the following:

- **Few nucleon reactions**
  \[ p(d,pp)n \]
  The BOL measurement typically lead to data not only in the regions where two-body processes dominate, but also outside of the reaction plane, where hardly any earlier experiment is available. FSI and QFI where compared with model calculations, also a Fadeev one. Interference effects were studied.
  \[ p(^3He,pd)p, d(d,pd)n, p(^3He,pp)d \]
  Strong contributions from quasi two-body reactions (FSI, QFS) were found. Discrepancies from model calculations appear especially in the interference region.

- **Break up of \(^3\text{He}\) on various nuclei:** \(^9\text{Be}, ^{12}\text{C}, ^{27}\text{Al}, ^{197}\text{Au}\).
  The latter reaction \(^{197}\text{Au}(^3\text{He},pd)^{197}\text{Au}\) shows a preference for elastic break up leaving the gold nucleus in its ground state. A strong directional correlation between \(p\) and \(d\) is observed. Use is made in the experiment of so called "local coincidences" i.e. coincidences within the detector.

- **Sequential reactions like \((\alpha,2\alpha)\).**

- **Particle unstable nuclear systems, like \(p^2, (pd), ^5\text{Li}, ^8\text{Be}, ^{12}\text{C^*}\).**
For instance in the latter experiment $^{12}\text{C}(d,d')^{12}\text{C}$, $^{12}\text{C} + ^8\text{Be}(g.s) + \alpha$, the angular correlations have been studied in and out of the reaction plane, and have been compared to model calculations.

4. WHAT DOES THE COMPUTER CONFIGURATION LOOK LIKE

The PDP8 normally used for data taking is a 4k, 12b, 1.5 μs machine with teletype, two magtapes (556 bpi, 30 ips), a clock, a storage scope and a keyboard (fig. 3). Naturally it is connected to the measuring electronics. It also has a memory to memory link with the X8. This link under X8 program control can be switched to either of the two PDP's. The other PDP has got 8k of memory, a teletype, a tally papertape punch and read, a magnetic disk (64k), and four video terminals. The experiment can be plugged into a somewhat simplified I/O buffer, connected to this PDP, a facility which is used only when the other PDP is down at a critical moment. The X8 is a 27 bits, 48k, 2.5 μs machine. It has got four telex-consoles, 5 magnetic tape units (800 bpi, 150 ips) a magnetic drum (512k, 21 ms) a 1000 ch/s paper tape reader, a 150 ch/s punch and a 10 lines/s line printer. The tape unit interfaces of the PDP are home-made. They are normal "IBM-compatible" from a tape point of view extended with a so called scatter read facility. This means, that one block, during reading can be sent to non consecutive memory parts. It is used f.i. when reading a program from tape which is longer than one memory page. An analogous facility is used in the interface to the X8, but here also a gather-write is possible. The PDP-station serves also as a data display terminal for the X8.

Something about the software

The X8 is used in time-sharing, the system being called WANNES. Experimental data are handled with the so called WINDOW-system (figs. 4,5). In the first place the latter system provides a high-level "language" for I/O handling: all
I/O channels look the same to programs in this language. In the second place it provides an easy way to deal with the hierarchically organized data at different levels (viz. appendix 1). The experiment-PDP is equipped with a similar, but of course much simplified version of the window idea under supervision of a monitor. Most data taking and primary data handling programs are standard with convenient control options. The other PDP is used for other then BOL tasks, mostly in cooperation with the X8.

A conversational symbolic programming language called SIMPLEX, somewhat analogous to ALGOL60 has been incorporated into the window system. This combination proved to be very effective for convenient data handling: the simple I/O and hierarchic handling of the window system together with the simple computational solutions in SIMPLEX.

An other important software part is formed by all kinds of editing programs and magtape program library handling programs.

5. HOW IS IT USED IN EXPERIMENTS

The second thing to do in an experiment, the first being a gross check of all system parts, is to focus the beam on the center of the target, with the help of several magnets. One can do it by looking on a fluorescence screen or what so ever, but the required measuring accuracy is $1^0$ implying an accuracy in position of a few tenth of one mm and of an accuracy in direction of also a few tenth of a degree. An obvious solution is to do an experiment which is sensitive to these parameters and which will take only a short time, so that measuring and adjustment can be iterated. So one takes f.e. a hydrogen target $(\text{CH}_2)$ and bombards it with the beam, looking for elastic coincidences between beam particles and protons. This measurement is "more than complete" with regard to the kinematic variables. By measuring with combinations of detectors in perpendicular planes one can calculate the beam position. This experiment is performed on-line with the X8, which takes data via the PDP and evaluates the position.
Normally the gross adjustment measurement takes a few minutes, while the fine adjustment, depending on required accuracy takes 15 to 30 minutes. The incoming events in this case are stored on drum. The physicist has a X8-telex available near his experiment operating desk, where he gives his commands and gets his answers.

Next in the experiment comes data taking. The X8 is not used in this stage. The PDP takes data, stores them on mag-tape, and prints a log on a teletype. One tape-take may last from 10 minutes to several hours. A run lasts for about one week, typically yielding a hundred tapes. Depending on the experiment the PDP may discard data but normally all incoming events are stored, i.e. 72 bits per particle, one particle for singles, two for two-fold coincidences, more for higher-fold coincidences. Singles and coincidences are measured separately in time, and are recorded on different tapes. Since commonly we intend to measure coincidences, only a minor fraction of singles tapes is recorded for later checks on several system parameters.

The following stage in an experiment is to calibrate energy scales and for all events convert ADC-output values into energy-values, to convert energy and energy-loss into energy and particle-type, to convert detector and position values into angles and to regather coincidences of one and the same reaction-event. In this stage the X8 is used intensively, reading and writing tapes, plotting spectra on the line-printer all in close conversation with the experimentator at the telex. The drum now is used for histogramming of data, which then can be visually inspected via the line-printer or PDP-storage scope. So here the display function of the PDP enters the game, the X8 having no scope display of its own, and it will play this role all along the data handling.

The step next normally is one of selection. The experimenter chooses particular particles, reaction Q-values and other parameters, in fact, a particular reaction process or class of processes. Note that all data are still available,
so that even if the chosen process was the pertinent aim of the experiment, later on other processes which occurred at the same time, can still be evaluated when interest for them arises after interpretation of the originally desired data. This possibility is used quite often and is one of the very advantages of BOL. In a way the beam plus target reactions are on tape like high energy physicists have them on photographs.

Now more and more computations enter further data handling such as center-of-mass transformations, and again the drum is used for histogramming. One important thing has to be done yet, namely the normalization for detection efficiency. Several procedures have been used to do it, one using a montecarlo technique: in the center of mass weighed random events are generated and transformed to the lab system according to the kinematics of the selected process. Now the detection sensitive area is taken into account and the resulting data are stored on magtape, thus simulating an experiment. So these data, when handled like experimental ones, yield the phase space distribution of the process, corrected for detection-geometry. The CERN random generator is used in this program. When all this has been done, the next step is to collect the data in physical interpretable histograms f.e. a Dalitz plot (fig. 6) and eventually compare the results with model calculations. Now the scope display is used intensively together with a hardcopy output when one is satisfied. The programs to perform the above things commonly are written in SIMPLEX, while several standard programs can be used such as a program to display pictures in perspective under variable looking angle and distance.

6. HOW DID IT GROW HISTORICALLY?

When in 1965 BOL was concepted kicksorters where still at the top. So we looked into a "very special" option available for our very modern 4k kicksorter, i.e. a buffer unit which could dump the kicksorter memory on magtape. Then we heard about some new small cheap computer to be used as a
programmable kicksorter, the PDP8. For both solutions we needed of course a larger computer to analyze the data.

The PDP was equipped with two cheap magtape units. To save money and to get into the field of interfacing we decided to design and to build the tape interfaces ourselves, which also would allow us to tailor it to our wishes! Now 4k of still available hardwired kicksorter memory can not be left unused by a physicist, so this memory would function as an autonomous display unit memory. A "highbrow" interface to the PDP was build complete with "jump" instructions to reduce flicker, light pen, top view and isometric display, with variable angle and light intensity variation. A memory to memory connection with the X8 was designed by us, and when the second PDP8 came in, certainly it had to be connected to the other PDP8. Still later the X8 connection was changed into a switch for both PDP's. The last PDP would serve more as a separate display unit, since the other would be rather busy with data taking. So a storage scope and hardcopy unit came in. The first interface boards where rather special purpose, but right now the digital group, born in meantime, has its convenient standard boards and methods.

Parallel with hardware the software developed. The window system for the X8 first, together with a simplified version for the PDP8. The X8 manufacturer delivered a time sharing system, but not at the first hardware delivery. For the PDP8 a more simple monitoring system was written by ourselves. Lots of editing programs were made and the window system was growing. A software group had been born, which a.o. remade the time-sharing system and then X8-use really started, experimental tapes coming in now and then already. Several standard data-handling programs had been written within the window-system and SIMPLEX was born. A three structured histogramming program was implemented. Of course many more important developments, could be mentioned, but let me conclude this section by telling what we use with satisfaction and what not. Not used are: the highbrow hardwired display unit, the inter PDP coupler. The tape interfaces were re-
designed even before testing. In the software several "standard" data handling programs are dying. But really all the rest is used satisfactory, though of course many minor changes in both hard- and software have been incorporated.

7. SOME IDEAS OF BUILDERS AND USERS NOW

The systems philosophy of BOL could be allegorized as: record everything now, analyze it later. Among others, this lead to a rather fixed hardware part including for the sake of argument the measuring electronics in the hardware, which for some experiments is felt to give too much non-specific data handling. But it also enables immediate measuring in regions of space where one is not sure a priori that it is worthwhile, and it has shown worthwhile in several instances. Also the simultaneous measurement of different processes leads to relative normalization, independent of beam current. And the kinematical completeness of measurements can be regarded as a virtue by itself. In contrast to the hardware rigidity, the software is rather flexible: in one program, on a high level one can deal with I/O problems and calculational tasks but one can also easily insert assembler code parts. These possibilities are fruitfully applied in program building and especially in non-standard data handling programming. But for standard data handling the very flexibility may become a burden since it consumes time. Time is felt as a problem through reliability also: for each run tests and calibrations have to be performed and some people regard the elapsed time between recording and first physical data output almost too long keeping involved sufficiently. But this seems not to hold for BOL only.

Though man-machine interaction is rather satisfactory (almost all programs are conversational via telex) the interaction could still be improved. Most of us agree on the success of some things: the partitioning of tasks between X8 and PDP's, the time sharing system of the X8, the great number of editing facilities, the common way to deal with data
in general, the I/O solution, the display station, the drum as a huge kicksorter, the array handling programs, and as a whole the possibility to perform complete measurements over a wide field of parameter values in many dimensions. An analogous discussion could be given about the non-computer parts of the measuring instrument.

8. **SOME CONCLUSIONS**

Building of the BOL system was a learning process to most of us and it certainly brought know-how into the Institute. Now after building and after some years of usage the following points may be important enough to be mentioned as a sort of personal conclusion.

- A time sharing system speeds up program development tremendously.
- So do editing programs.
- Modularity is an aid to optimize the system. This holds for measuring mechanics and electronics, computer hardware, and computer software.
- Convenient standard building blocks on different levels are an aid to modularity. A convenient degree of modularity, I guess can only be found by iteration.
- Users experience and feedback are other aids to optimize the system. Modularity helps feedback through getting users experience on parts of the system and through making changes easy.
- Feedback may help to make the choice between a general or a specific data handling solution.
- Man-machine interaction should be strongly man oriented. This has been said so often, that repetition seems superfluous. I think the essential problem is, that just how strong "strong" is to be can only be grasped by experience. Shortly, though we know BOL can still be optimized, we feel that it is useful for the purpose it was intended.
English publications of "technical" data on BOL are:


6) Test procedures for the multidetector BOL-system, M.A.A. Sonnemans et al., Nucl. Instr. & Meth. 92 (1971) 193.

7) Data acquisition with the BOL nuclear detection system, R. van Dantzig et al., Nucl. Instr. & Meth. 92 (1971) 199.


9) Coincidence measurements on break-up of few nucleon systems with the multidetector BOL, R. van Dantzig, doctor's thesis 1971.


Papers at the conference "Few particle problems in nuclear reactions", (Los Angeles, August 1972):

13) Three-body reactions studied with "BOL", R. van Dantzig et al.

14) Angular distributions for the reactions $^3$He+p$\to d+(pp)$ and $^3$He+p$\to p+(pd)$, B.J. Wielinga et al.

15) Treiman-Yang test for d$\to d+p+n$, A.D. IJpennberg et al.

16) Peripheral break-up of $^3$He on $^{197}$Au, W. Hermsen et al.

17) Experimental tests on the purity of nucleon $^1$S final-state interaction, R. van Dantzig et al.

18) A study of the reaction p(d,pp)n, B.J. Wielinga et al.

APPENDIX 1

In fig. 5A, the name "INFO" in the macro "IN" can be any of the following flows:

<table>
<thead>
<tr>
<th>name</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX</td>
<td>from mag.tape to X8-memory</td>
</tr>
<tr>
<td>AX</td>
<td>from drum-array to X8-memory</td>
</tr>
<tr>
<td>BX</td>
<td>from paper-tape to X8-memory</td>
</tr>
<tr>
<td>PX</td>
<td>from PDP8 to X8-memory</td>
</tr>
</tbody>
</table>

Similarly the "OUTFO" in OUTPUT can be:

<table>
<thead>
<tr>
<th>name</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>XT</td>
<td>from X8-memory to mag.tape</td>
</tr>
<tr>
<td>XA</td>
<td>from X8-memory to drum-array</td>
</tr>
<tr>
<td>XB</td>
<td>from X8-memory to paper tape</td>
</tr>
<tr>
<td>XR</td>
<td>from X8-memory to line printer (several formats a.o. tabel, topview)</td>
</tr>
<tr>
<td>XP</td>
<td>from X8-memory to PDP8.</td>
</tr>
</tbody>
</table>

The name "IR" is for IKO-Rekord. It is the smallest info-quantum in the hierarchy:

IR, BLOCK, TRAIN, NET, GROUP, FILE, REPFILE

Commonly it is a 72-bits experiment data word, but it can be anything else. The structure of an IR is defined by a so-called "chord" definition, specifying a sort of variable byte-structure: Example:

1 "identifier"
3 "number of X8-words"
5;5;1;1;6;6; "number of bits/byte; idem..., word 1
12;3;9; "number of bits/byte; idem..., word 2
3;9;1;1;1;1;1;7; "number of bits/byte; idem..., word 3

This is the structure of the experiment data word in the X8, two PDP8-words (fig.4) being packed into one X8 word.

For drum arrays the following hierarchy exists:

GROUP, ARRAY, ROW, COLUMN, AR

AR, Array-Rekord, being the lowest quantum.

The drum, used as a huge kicksorter, can be devided according to the so called group-definitions.

There are four classes of groups with the following group-
numbers

group   function

class 1:  0-31: histogramming in full word length: 24 bits
         32-63: histogramming in $\frac{1}{4}$ word length: 12 bits
         64-95: histogramming in $\frac{1}{4}$ word length: 6 bits
         96-127: histogramming in $\frac{1}{8}$ word length: 3 bits

Example:
3       "number of groups to be defined"
0; 5; 1; 4096  "group identifier, number of arrays, number of rows, number of columns (drumspace: 20k)"
32; 5; 1; 4096  "etc. (drumspace: 10k)
72; 64; 64; 64  "etc. (drumspace: 64k)"

This work is part of the research program of the Institute for Nuclear Physics Research (I.K.O.), made possible by financial support from the Foundation for Fundamental Research on M (F.O.M.) and the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).
Fig. 1  Upper: the BOL scattering chamber with the tons containing ADC-boards at left, beam entering from front, most detection units inserted.
Lower left: operating panel, DEC PDP8 and display terminal.
Lower right: EL-X8 computer.
 Beamparticles: d, p, \(^{4}\)He, \(^{3}\)He
Energy (MeV): 23-28, 48-58, 48-58, 62-75
Current extern I (nA): 200, 60, 20, 20
Current in BOL: I x \(10^{-3}\); burst: \(v = 1500\text{s}^{-1}\)

5 Quadrupole-, 3 horizontal and 3 vertical bending magnets, 3 slits
6 Fluorescence screens, optical alignment, communicating liquid level system

3 Concentric spheres: aligning and cooling detectors- vacuum, aircool. electron.
64 holes for charged part. detection units: 13.2% of 4π,
angular resolution \(\theta^O\): 10 holes for \(\gamma\) detection units
Vacuum (sublimator ion getter pumps): 10\(^{-6}\) Torr
10-target system: freon expansion cooling: -25\(^\circ\)

64 Checkerboard detectors: 0.3 mm thick, 88 fields, 20-55 keV noise
\(\sim\)64 Si(Li) p-i-n detectors: 5 mm thick, < 50 keV noise, 0.2 mm backside window
\(\sim\)128 Si(Li) p-i-n detectors: 3.6 mm thick, 0.03 mm backside window
(thin surface barrier detectors: 0.010 mm thick)
(Ge(Li) detectors)

Paraday cup
Position signals 30 ns: Charge-sensitive ampl., clip 1.25 \(\mu\)s
Coincidence system: 80 ns, remote control multiplicity
Logic: signal - CU, position sign., start ADC, store 72-bit register
start sequencer (0.3 \(\mu\)s per channel, 2.5 \(\mu\)s readout), NKE

64 ADC's: 8192 channels, 100 Mc, 100 \(\mu\)s; 192 ADC's: 512 chan., 6 Mc.
72-bits register

![Diagram of the BOL system](image)

PDP 8 (1) : 4 k, 12 bits, 1.5 \(\mu\)s memory; Peripherals:
[2 DATAMAG, 7070 tape units]
[4 k, 18 bits, 20 \(\mu\)s memory-scope]
Interfaces to PDP8(2) and EL-X8, etc.

PDP8(2):
[4 k, 12 bits, 1.5 \(\mu\)s memory; Peripherals:
Storage/display
Interfaces to PDP8(1) and EL-X8, etc.

EL-X8:
[48 k, 27 bits, 2.5 \(\mu\)s memory; Peripherals:
Drum: 512 k, 27 bit words
Line printer (20/20 lines per second)
4 Magnetic tape units (CDC)
Paper tape punch/reader
Interfaces to PDP8(1) and PDP8(2)

MONIKOR: Multiprogramming
monochrome, non-system with modular asynchronous programs running non-interferingly in parallel

VAX/8: Time sharing system. Multiple access, file oriented, swapping mechanic and flexible memory allocation system for large memory.

Data analysis with WINDOW-processing: Independent I/O programming/Transformation/Counting on drumstore upon selection/representation on lineprinter and storage display/Level structured programming.

Fig. 2 Schematic survey of the BOL system
Fig. 3 Survey of the BOL computer configuration
### Checkerboard code

<table>
<thead>
<tr>
<th>Position code</th>
<th>Left code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gold side</td>
<td>Gold</td>
</tr>
<tr>
<td>5 bits</td>
<td>1 bit</td>
</tr>
<tr>
<td>Aluminium side</td>
<td>Aluminium</td>
</tr>
<tr>
<td>5 bits</td>
<td>1 bit</td>
</tr>
</tbody>
</table>

- **Spare ADC:** 6 bits
- **Detection channel number:** 6 bits
- **Sum energy E (bits 0-11):** 12 bits
- **Coincidence multiplicity:** 3 bits
- **Checkerboard energy DE:** 9 bits
- **Coincidence group mark:** 3 bits
- **Spare ADC:** 9 bits

<table>
<thead>
<tr>
<th>Overflow E</th>
<th>13th bit E</th>
<th>Overflow DE</th>
<th>Pile-up</th>
<th>Overflow NME</th>
<th>NME (dead time scaler)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>1 bit</td>
<td>1 bit</td>
<td>1 bit</td>
<td>1 bit</td>
<td>number of measured events</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7 bits</td>
</tr>
</tbody>
</table>

**Fig. 4** Construction of a 72-bits data word, the most raw form of stored information.
Fig. 5  
A: an example of a simple WINDOW program  
B: part of a function detection in SIMPLEX  
C: SIMPLEX part executed within a window program to calculate center of mass energy for each particle
Fig. 6 Dalitz spectrum, a highly analyzed form of the measured data
RADIO TELESCOPE CONTROL

J. Schraml

Max-Planck-Institut für Radioastronomie
Bonn, Germany

INTRODUCTION

The history of the application of computers in radio astronomy is very similar to that in other fields of experimental science. At first computers were employed in data analysis and reduction and as confidence grew, they were used to monitor and control parts of the equipment. More sophisticated instruments were planned and the tight specifications could only be met by designing these instruments with on-line computer control. Once the computer was available the next logical step was to schedule and supervise the whole observing procedure according to the wishes and needs of the astronomer with the aid of the computer.

In this lecture I want to report on such a control process. It has been developed for the 100-m radio telescope of the Max-Planck-Institut für Radioastronomie in Bonn. The instrument is the largest fully steerable antenna in the world, located next to the village of Effelsberg some 30 km SW of Bonn. Its operation started on May 31st this year. It is controlled by a Ferranti Argus 500 on-line computer. The lecture is divided into three topics. The first one deals with the process itself, the radio telescope and its operation, and the demands resulting for the control program. The second part briefly describes the computer and its hardware. The third one introduces the architecture of the executive program in general, which has been tailored to meet the demands of the process and the hardware. The communication
between the observer and the system, the format of data on magnetic tape and an on-line reduction of position measurements, are dealt with in somewhat more detail.

1. **THE RADIO TELESCOPE**

The earth's atmosphere is transparent to electromagnetic waves from about 10 MHz up to 300 GHz or 30 m down to 1 mm wavelength. Through this "radio window" astronomers investigate the radiation originating in celestial objects with radio telescopes. A radio telescope system consists of two parts:

a) the receiver, that analyses intensity and spectral properties of the incoming radiation, which is collected by

b) the antenna from the direction under investigation.

1.1 The receiver

The objects observed in radio astronomy are in general not the stars, but rather the emission of huge clouds of gas of extremely low density. The signals received cannot be distinguished from statistical fluctuations arising in the receiver itself. In order to reduce these unwanted contributions to a minimum, low noise devices like cooled parametric amplifiers are used as first stages of amplification. They are installed as close to the focus as feasible. This part of the receiver is called "front-end" and a typical arrangement is shown in a schematic diagram in Figure 1. The amplifier can be connected either to the antenna or to a reference termination. The latter is a resistor held at a constant temperature $T_r$ and the statistical fluctuations or noise equivalent power emitted by it is proportional to $T_r$. The signals from the antenna can thus be calibrated this way and are therefore always given in antenna temperatures, $T_A$. The noise contribution of the receiver is added to both of these routes and Figure 2 gives an example of the output of
the receiver. The minimum signal that can be detected is given by

\[ <T_{\text{min}} > = \frac{T_{\text{RX}}}{\sqrt{B \cdot \tau}} \]

where \( T_{\text{RX}} \) is the noise equivalent temperature of the receiver, \( B \) the analyzing bandwidth and \( \tau \) the integration time. For a given receiver and type of measurement, the first two quantities are fixed; then only an increase in integration time will yield more sensitivity. This is only true as long as the receiver level is constant. But in practise there exist long term instabilities as also shown in Figure 2. A technique to minimize drift problems is to use switched receivers as proposed by Dicke. In the front-end of Figure 1 the amplifier is switched periodically between both ports of \( S_1 \) and the difference is formed

\[ \text{Signal} = (T_A + T_{\text{RX}}) - (T_R + T_{\text{RX}}) \]

If the Dicke switch rate is much shorter than the time scale of the instabilities, one obtains the necessary long term stability at the expense of losing half of the observing time. The Dicke switch also connects in turn two integrators to the detector synchronously and these are read by the
computer. The difference is formed, integrated over a number of basic cycles as specified by the observer and displayed on a chart record. This is a simplified version of a broadband "back-end", the data rate is governed by the Dicke rate which is usually around 10 to 100 Hz.

For the 100-m telescope this back-end has been extended to allow for continuous gain calibration. At half the Dicke rate a calibration signal is added via $S_2$ and four phases read by the Argus, i.e. antenna, reference, antenna + CAL, and reference + CAL. By the proper arithmetic the calibration and signal can be extracted. A modified procedure might be possible in quite different types of experiments and has three major advantages.

a) no observing time is lost due to calibration
b) the integration time on the signal is about the same as on the calibration
c) nonlinearities in the equipment can be detected and corrected for.

For spectral line work the whole bandwidth of the receiver is split up into a number of channels. For this purpose a 384 channel autocorrelation back-end has been built consisting of a 384 bit shift register and 384 correlators and integrators. In intervals specified by the observer the integrators are read into a DDP 516 and a fast Fourier transform is performed on-line to yield the spectral distribution. At present the results of the transform are transferred word by word by request into the Argus. There position information is added and the data written on tape. A two way data link is now being tested, which transfers data via DMA and has an optional packing of the most significant 24 bits of two DDP 16 bit words into one 24 bit Argus word.

For pulsar measurements a 1024 channel pulse-height analyser is triggered with the period of the pulsar and
integrated for a specified number of cycles. The integrators are 24 bit long, the data stored in BCD. They are transferred the same way as the spectral data, but then converted into binary in order to save the conversion time for the off-line CDC 3300 in Bonn.

1.2 The Antenna

The most versatile antenna type is a parabolic reflector. The 100-m antenna is of that kind and shown in Figure 3. Its large main parabolic reflector has a diameter of 100 m and a focal length of 30 m. The reflecting surface consists of pre-shaped aluminum panels arranged in 17 concentric rings of which the outer 4 have a perforated surface. The reflector can be tilted around a massive elevation axis with a maximum speed of 20°/min. The axis is supported by 2 Λ-shaped towers of 50 m height. There the position of the axis is read out by an encoder mounted on each end. The Λ-tower rest on 4 drive-trains that can rotate the whole construction in azimuth around a track of 65 m in diameter with a maximum rate of 40°/min. This track has to support the total weight of 3300 tons and determines the accuracy of the level of the elevation axis. The antenna is held on the track by a bearing at the end of the cylinder in the center. There the azimuth encoder is mounted. All cables needed in the rotating and tilting part hang down in that cylinder and can be twisted over its length so that a total rotation of 400° is possible. This imposes a topological problem on the on-line program. It has to unwind the cables before and not during the observation. Supported by four legs an elliptical subreflector of 6.5 m diameter and 2 m focal length is mounted confocally to the primary dish. It will reflect the incoming radiation again and focus it a second time in the apex of the main dish. This arrangement of reflectors is called a Gregory system. Our first receivers will be used in the prime focus. For this purpose the front-end is
Fig. 3 The 100 m radio telescope at Effelsberg
mounted in a 2 m long tube that sticks out through a hole in the center of the sub-reflector. The whole secondary mount can be shifted in elevation, moved axially towards and away from the main dish, and rotated to allow focussing and rotation of the plane of polarization. All these motions will be controlled by the Argus according to the requirements resulting from the behaviour of the steel construction and the astronomical purpose.

Two properties of antennas are of interest to the astronomer: the sensitivity or collecting area and the directivity or angular resolution.

The signal strength reflected into the receiver is proportional to the effective collecting area. Unfortunately the effective area $A_e$ is always smaller than the physical area $A$ of the antenna. The feed supports block part of the aperture and the illumination of the reflector has to taper off towards the rim in order to avoid spillover of ground radiation. Figure 4 shows the gain of the antenna as a function of frequency. When the wavelength approaches the dimensions of the roughness of the surface, the reflection becomes diffuse and the gain drops. The kink occurs at a wavelength of about 10 times the surface accuracy. Fig. 4 Antenna gain

Astronomers want to use the 100-m telescope at centimeter wavelengths. Therefore the shape of the parabola has to be retained to better than 2 mm under observing conditions. If one tries to make such a rigid structure it will become very heavy and expensive and one might reach the limits of the elasticity of steel. But deformations due to
gravity do not change the performance when the deformed instrument will be another parabola. The 100-m antenna is the first one that is designed to deform homologously. When the reflector is being tilted, it folds and the focal length becomes shorter. In addition the upper part of the rim sags which results in a new apex. The on-line computer has constantly to refocus and realign the electrical axis. In moving from the zenith to 7° above horizon, the outer panels move as much as 10 cm with respect to each other while the surface nevertheless remains a parabola to within 2 mm.

Figure 5 shows the output of the receiver when the antenna is scanned through a point source along one coordinate. The output can be well approximated by a gaussian distribution centered around the position of the source. The width at half signal level is called half power beam width 0 and is a measure of the resolution of the antenna. The angular resolution of a telescope is proportional to the ratio of wavelength to aperture. Typical values for the 100-m antenna at various wavelength are:

\[
\begin{align*}
\lambda &= 11 \text{ cm} & 0 &= 415 \\
\lambda &= 6 \text{ cm} & 0 &= 214 \\
\lambda &= 1 \text{ cm} & 0 &= 40''
\end{align*}
\]

For accurate intensity measurements the antenna has to be kept on the top of the response, i.e. within about a tenth of the beam width. At 1 cm this corresponds to a mere 4'' or 1 mm at the azimuth motors or .5 mm at the elevation ones. The position of the axes can be read in increments of 2''. To
make full use of this accuracy the commanded positions have to be calculated to 1" and all astronomical effects of this magnitude have to be included.

For terrestrial objects both coordinates remain constant throughout an observation. But for astronomical observations it is necessary to compensate for the daily rotation of the earth. If one axis of the telescope is mounted parallel to the earth's axis the reflector could be rotated around this axis at a constant rate of $2\pi/23^{h}56^{m}$ and the other coordinate would remain constant. But this would require immense technical effort in the construction of the telescope mount. Therefore the on-line computer is used to transform this constant motion into the axes of the horizontally mounted instrument. This rotation is fully determined by time and the geographical location of the instrument and results in an accelerated motion in both coordinates.

The astronomer still would have to input the apparent position of the object for the day of observation. This position changes due to the precession of the earth's axis with 50"/year, the varying rate due to the elliptical orbit of the earth and a parallax due to the 30 km/sec velocity of the earth around the sun, all of which is dependent on the calendar date. As shown in Figure 6 a much more general approach to the astronomical concept was developed by P. Stumpff and myself. The observer can orient his coordinate system with regard to the geometry of his object by specifying 3 angles and the program "freezes" the telescope to the object. Then the motions in this system are very simple. There exist no electronic counterpart to a photographic plate, so extended objects have to be observed point by point by moving the antenna relative to the source parallel to the major axes. If the observer is interested in the properties of the instrument he will use the source as signal generator and move parallel to the instrumental axes. In
BASIC COORDINATES \( (l, b) \)
ARE THOSE WHICH THE
OBSERVER HAS AVAIL-
ABLE IN A CATALOGUE
OR IN A MAP
FOR EXAMPLE:
\[ l = \delta 50, \quad b = \beta 50 \]
OR:
\[ l = l_\Pi, \quad b = b_\Pi \]
WITH REGARD TO THE
BASIC SYSTEM, THE
DESCRIPTIVE SYSTEM
\( (\lambda, \beta) \) IS DEFINED
BY THE THREE
NUMBERS:
\[ l_\Pi, b_\Pi, \kappa \]
DESCRIPTIVE COORDINATES
\( (\lambda, \beta) \) MAY BE CENTERED
AT A RADIO SOURCE AND
ORIENTATED WITH REGARD
TO GEOMETRICAL STRUCTU-
RES WITHIN THE SOURCE.

CONTOUR MAPS: SCANNING IN \( \lambda \) OR \( \beta \)

\[ \lambda(t) = \lambda_0 + \lambda s(t - t_0) \]
\[ \beta(t) = \beta_0 + \beta s(t - t_0) \]

INTEGRATION: ON-OFF'S

ON: \( \lambda = 0, \beta = 0 \)
OFF: \( \lambda = \lambda_0, \beta = \beta_0 \)

POINTING, REFRACTION, etc.

HORIZON

Fig. 6
Fig. 7 Logical flow chart of astronomical program
Figure 7 the flow chart of the astronomical program is given. From the observer input and a clock and calendar the mean position and velocities for the time of the observations are calculated, the parallax correction added (APPARAD) and the telescope position and velocities calculated. These positions are corrected for zero errors of the encoders and misalignment of the axes of the telescope to yield the commanded position. The pointing corrections have been determined in a test period by observing sources of known position. It is the task of the servo to slave the telescope to the commanded position. From the commanded position and the indicated ones the servo signal is generated. It controls the velocity of the motors. In an earlier optimization period the amplification factors for the three feedback loops have been determined. In order to reach a large dynamic range in possible velocities, gears have to be shifted. For slewing the anti-backlash braking of half the motors must be removed. All these control functions have to be initiated and supervised by the control program. The fundamental resonant frequencies of the telescope are between 1 and 2.5 Hz, and this requires that the output to the servo is updated with 20 Hz or every 50 ms.

1.3 Operation

Once a particular receiver is installed the telescope is operated continuously. The time is scheduled on an hourly basis among observers with a very wide range of technical understanding. They all use the same equipment in various modes and the change over should not introduce any other delay than the slewing time between the objects. No one can test his equipment or program prior to the observation without interfering with other observers. On the average our staff members will get some 40 hours of observing time per year. This means that we had to provide a self contained system which guarantees the observer that the telescope is
observing his region of interest, that data are recorded with all necessary information and the function of the equipment is checked continuously and displayed in a way to be understood easily. This might be quite time and space consuming. Many items are buffered several times to be certain that the exact time relations between receiver and telescope data can be guaranteed. The encoder or the clock reading programs for instance might take up to ten times as long as the mere read instructions, since the readout is of vital importance and is checked in various ways if there might be faults and if they can be corrected in real time. Since the final data reduction is done in Bonn, we felt that half of the time and core available can be devoted to the process control.

2. **THE ON-LINE COMPUTER**

The Ferranti Argus 500 has a wordlength of 24 bits and a cycle time of 1.1 μs. The wordlength is quite advantageous for the telescope control, since 1" corresponds to 20 bits. Hence none of the calculations have to be performed in double precision. The same is true if receiver data are to be integrated over longer periods of time. And it allows addressing of pages of 8 k in the instruction. The instruction format is shown below.

```
X F M R [ Adress ]
```

It has 32 functions that operate on 7 registers which are actually the first words of the memory. The 3 first ones can be used for address modification. The 14th bit of the address (R) is replaced by a three bit page register and there are 3 kinds of relativisers, one for reading, one for writing and one for the order number.

The machine has a very powerful interrupt structure and
allows some hardware protection through it. It has 8 sets of registers and an additional 8 working cells and the set that is used can be selected by the organizer by setting a \textit{X relativiser} for the registers. Many subroutines can be written reentrant for different levels by keeping intermediate results in the working space. Once a level other than 0 is selected, the read, write and \textit{X relativisers} cannot be changed. All hardware interrupts set the order number to the same address and reset the \textit{X relativiser} back to 0 for the organizer.

One of the instructions is an exchange instruction. This is a true multi-programming instruction. It allows the transferal of data between levels of different priorities without having to set lockout since the interrupt can only occur before or after the instruction.

The Argus 500 at Effelsberg has 24 k of memory that are arranged as shown in the following diagram.

![Diagram of memory layout](image)

The addresses \(< 10000\) are I/O addresses. Block 0 contains the organizer, subroutines that are accessible to all blocks.
like mathematical routines and I/O routines, and all current receiver and antenna data. For the positions also their trigonometrical functions are stored and available to all programs. The programs used for the observations are in block 1. Block 2 is only 4 k long and is used nearly entirely for buffer space to write blocked data onto magnetic tape. Block 3 is available for installing user programs later.

The Argus is equipped with a 600 k fixed head disc, a magnetic tape unit with 3 decks, an X-Y plotter, a line printer, 2 typewriters, 2 teletypes and a alphanumeric display. Most of digital input and output cards are rather slow, since the computer was ordered in 1967. Therefore it takes 160 ms to read in all the 384 relays or switches that give information about the state of the gears, the focus, operators intervention and safety switches.

3. THE CONTROL PROGRAM

The organization of the control program is strictly time oriented and reflects simply the requirement listed thus far. The program has four different levels of priority which also run on different levels of the X relativisers and thus can be interrupted without problems. Except for the Calcomp the only interrupt used is the 10 ms time interrupt.

At the level of the organizer the I/O equipment is served. Since the Dicke rate is asynchronous to the computer clock the integrators are tested each time and receiver data read into buffers when necessary.

The next level of priority is the 50 ms telescope cycle. It only can be interrupted by the plotter. The main effort of program development went into making this part very efficient. When a new observing program is started by the operator all time dependent astronomical quantities and
their derivatives are calculated for the start time in the background. The program is activated when this preparation is finished, and a linear extrapolation can be applied. Another way to speed up calculations was to linearize nested loops and save intermediate results. The task has been split into 5 parts of about equal length, and every 10 ms the next one is activated immediately after the level 0 program. Thus the time load is distributed about evenly between the interrupts. When this level is suspended the program that was interrupted is continued if it was from the next level of priorities or the present X relativiser level is suspended and the next one is started if the program that was interrupted was of lowest priority. This allows a time slicing between the three low priority levels.

The programs at the third level are activated every 160 ms, since a full new set of outside information is then available. These programs include control and monitoring the gears and limit switches of the antenna, the focus control, the communication with the observer and operator as well as the magnetic tape buffering program and driver. These programs reassure operation after an interrupt until they suspend themselves because of waiting for I/O or having finished the task once.

The data transfer between the DDP and the Argus at present runs on a time sliced level, since hardware malfunctions sometimes have occupied the level for too long a period of time. On-line data reduction at the end of a scan shares the same level. And with time slicing it is possible to run an off-line program like assemblers or debugging programs without modification. No waiting for I/O can upset the system. In future we want to have a rigorous tracing program for block 3 of the memory run on one of the time sliced levels, so that observers can program their own on-line analysis in Fortran or a similar language and never
can interfere with their own observations. This will be simplified by the hardware protection features of the ARGUS. The computer time is calculated on one of these levels. The overhead with the I/O drivers is less than 10% and depending on the data rate and scan options 40 - 70% of the computer time is still available.

A trained operator is present throughout the operation and is responsible for the equipment. He decides if an observing program can be started or might have to be deleted. His decisions are signaled to the control program by means of switches.

![Diagram](image)

**Fig. 8**

The observer has to input much more detailed information. A special input language has been developed for that purpose. As shown in Figure 8 an observer can fill the instructions for the next observing program in a passive option array without disturbing the current program. He has plenty of time to check his input to the teletype, supplement it and correct mistakes. Only quantities have to be specified that
are different for the next program since a single passive field is used.

The program will be activated by the operator. The time he pushes the button will be $T_0$, the reference for all astronomical quantities. The necessary advance calculations are performed in the background, the passive field transferred in the unused one of two active fields. At the end of this transfer a pointer is changed to the new array. In this way there never exists an undefined state of the drive program.

The input language consists of a mnemonic four letter code followed by a numerical quantity, where necessary. The first of the four letters defines an operator, the next three select a coordinate or program function like the background to be used or number of repetitions. If the four letters stand for a valid command, the system types a colon, otherwise a question mark. The input format for the numerical values is fairly free of restrictions. As an example the instructions for scanning through a source along one coordinate are listed below.

\begin{verbatim}
SOTZ:
SEAS: 1
SLAY: 12 26 33.3 S
SBET: 2 19.7 '
OBET: -10 '
SBST: 27 30
VEBT: 5 '
SPPI: 6
\end{verbatim}

The first line clears all the offsets left by previous observers. In basis system 1 all coordinates are given for the standard epoch 1950.0. With the next two instructions the observer has specified the position $\alpha^5_0 = 12^h 26^m 33.3^s$ and $\delta^5_0 = 2^o 19.42'$. With these instruction alone the telescope would be pointed at the quasi-stellar object 3C 273. Relative to the peak the observer wants to start 10' south, scan for 27.5' north at a rate of 5'/min. One of these subscans would
thus last 5.5 minutes. The subscans will be repeated 6 times, 3 times going up and three times down along the same strip of the sky in order to save telescope travel time.

**OLAM**: 10 S

When the operator activates the program, the antenna slews to a position just south of the starting point, where it waits until the operator starts the scan. From then on the whole observation is performed automatically for the next 33 minutes and a couple of seconds that it takes to reverse the motion of the antenna. During this time the observer only has to type the last instruction. After a next activation the whole procedure would be repeated with 150° added to the previously constant coordinate.

If the ratio of the distance to velocity is not a multiple of the basic integration time, the telescope will continue to scan across the end position until the next integration cycle is completed. This is an example of a situation in which the receiver controls the motion of the antenna. An example of the opposite situation occurs in spectral line observations: The velocity of telescope (earth) in the direction of the source is computed on-line and the frequency of the local oscillator in the receiver is periodically updated to correct for the resulting Doppler shift.

In every automatic process it is important to have a manual override which can be put into operation without too much time delay in case of interference. Within 200 ms, the operator can interrupt a subscan and either restart it or terminate the whole observation. In either case, a flag is added to the data recorded immediately prior to the intervention.

Intimately connected with the automatic execution of the observing instruction is the recording of receiver and posi-
tion data on magnetic tape for further off-line reduction. From the programming point of view, the information to be recorded consist of 2 types: a header that contains quantities that are constant for a whole subscan like observer input, $T_0$, date; and, at the end of each integration cycle, a multidimensional data point consisting of the output from the receiver channels and other antenna readouts. The lengths of the headers and data points are different for each back-end in use and these quantities are written into the header. The header also contains a code that indicates which decoding subroutine is to be used at run time. This way, data from various backends can be mixed on the tape. The data points are added individually into buffers and so no limitation to the length of an observation is imposed.

In the buffer the data are blocked in multiples of 90 words. When 10 of these sub-blocks are filled they are transferred together to the magnetic tape. Between each sub-block, a CDC compatible counter is written that will be interpreted by the CDC system in Bonn. Thus, the deblocking is done automatically without increasing the length of the Fortran program. The observer does not see any of the block structure or the different representation of information on the two computers.

In order to write error free tapes, we have to copy the buffer to be written first into core, dump it onto tape, re-read it and compare it with the original and repeat the procedure with a different piece of the tape until the transfer was successful. The observations would be suspended if all the buffers were filled. Thus far, that has only happened once and was due to a faulty magnetic tape controller; the maintenance crew repaired it within hours. I have not heard of a single error on tape.

Methods of on-line data analysis can be quite different
from the procedures used for final reduction. Their purpose is to give quick reference about the status of the equipment and provide corrections that will improve the next set of measurements.

In continuum observations, the signals are recorded in equally spaced intervals; let us number them by $i$. In order to calculate the flux, or the area under the curve, we have only to sum the $T_i$. To determine the position of the center of the source the center of gravity or the first moment of the distribution is calculated. This requires a summation of $i \cdot T_i$. Both results might be biased by a slope in the baseline of the receiver and this slope should be removed. The first points of the baseline are obviously the first points of the scan. The baseline at the end is obtained by a cyclic storage of the $T_i$ which finally will yield the last points. A linear baseline is then interpolated and subtracted. This method has a special advantage for on-line analysis in that it does not require storage of all the data points in a subscan. The results output onto a typewriter at the end of a subscan and are very helpful in the determination of the pointing corrections.
CASE HISTORY: A HIGH-ENERGY PHYSICS EXPERIMENT

S. Conetti
CERN-Trieste High-Energy Group
CERN, Geneva, Switzerland
Istituto di Fisica dell'Università di Trieste, Italy
Istituto Nazionale di Fisica Nucleare, Sezione di Trieste, Italy

1. GENERALITIES

In a high-energy physics experiment, the task that the on-line computer is required to perform is a combination of data acquisition and process control. The source of data is the experimental apparatus where "events" are generated. From the computer's point of view, this results in the request for a data input to be executed with the minimum possible delay. Subsequent tasks are to process the recorded data in order to monitor their quality, and to transfer the raw data themselves and/or some derived quantity to a storage device. Concurrently the computer should be able to check the correct functioning of the apparatus, warning the experimenters of any faults, or even taking action to correct the fault. The results of the computer's supervision work must be continuously available to the experimenters in the form of histograms, tables, plots, etc.

Although the variety of activities requires the on-line software to have multiprogramming features, simplifications come from the fact that the types of occurrences of input/output requests are known a priori and are to be executed according to a well-defined scale of priorities. The various programs that share the core storage can be thought of as independent from each other, except for the sharing of some common blocks of data on which the programs operate (see Fig. 1). A scheduler will be responsible for the coordination of the different activities, applying the scheduling rules shown in Table 1, where the activities are listed in a possible order of decreasing priority. The relative magnitude and importance of the listed activities depend strongly on the characteristics of the experiment, the computer configuration and the hardware available, for example:
Fig. 1  Layout of the on-line computer main activities. The independent activities operate on the common storage blocks (represented by the thick-sided boxes).

- In a particular experiment with a very high event rate, the time and storage requirements for the data acquisition can be such as to prevent most of the remaining operations.

- The presence of a data link with a bigger computer will reduce the amount of analysis performed by the small computer, increasing in turn the work on data transfer.

- The degree of complexity attainable for the process control is dependent on the available instrumentation: a system like CAMAC\(^1\) can be very useful in reducing the number of input/output channels to the experiment and in standardizing the software needed to operate on a wide range of instruments.

In addition to those mentioned above, more activities could be included in the scheduler's agenda: compilations, assemblings, off-line (interactive) analysis of pre-recorded data could all be performed in the spare time. Moreover, some appropriate software could be present in order
<table>
<thead>
<tr>
<th>Activity</th>
<th>Scheduled by</th>
<th>Source of data</th>
<th>Data destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast data acquisition</td>
<td>High priority interrupt from experiment</td>
<td>Experiment</td>
<td>Raw data storage</td>
</tr>
<tr>
<td>Data transfer</td>
<td>Data acquisition and/or analysis</td>
<td>Raw data storage</td>
<td>Magnetic tape unit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cooked data storage</td>
<td>Data link</td>
</tr>
<tr>
<td>Analysis</td>
<td>Data acquisition</td>
<td>Raw data storage</td>
<td>Cooked data storage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Table, histograms</td>
</tr>
<tr>
<td>Process control</td>
<td>Internal clock interrupt</td>
<td>Experiment</td>
<td>Table, histograms</td>
</tr>
<tr>
<td></td>
<td>Software counter(s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interrupts from experiment</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(e.g. CAMAC LAM)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>User request</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interaction with users</td>
<td>User request</td>
<td>Tables, histograms</td>
<td>Teletype, Printer, Display, Plotter</td>
</tr>
</tbody>
</table>
to operate the computer as a desk calculator. To achieve such a high performance a disc becomes indispensable: the bigger requirements of core storage for an inevitably more complicated operating system are well rewarded. With the possibility of overlaying the programs, the use of high level languages turns out to be efficient, giving all the experimenters easier access to the computer. A greater flexibility is also reached in the experiment's setting-up stage, when in general the sequence of required operations is not defined a priori, in opposition to the data-taking stage when usually a single factotum program is responsible for all the operations.

2. **THE CERN-TRIESTE EXPERIMENT**

The aim of the experiment described here is to measure the polarization parameter for the reactions $\pi^+ p^+ \rightarrow p\pi^+$, $K^+ p^+ \rightarrow pK^+$ and $\pi^+ p^+ \rightarrow \Sigma^+ K^+$. A beam of positive $\pi^+$ and $K^+$ mesons hits a target (a sample of butanol) where the free protons are constrained to have their spins orientated in the same direction. Of all the possible reactions taking place when the beam hits the target, the interest of this particular experiment is in those reactions in which a pion (kaon) from the beam, colliding with a free proton, transfers to the target most of its energy so that the proton starts travelling forward and the meson scatters backwards. Also of interest are the events in which an exchange of quantum numbers is added to the described energy transfer, so that the outgoing particles are of a different type, a $\Sigma^+$ and a $K^+$. The experimental apparatus consists of a combination of scintillation and Čerenkov counters, as well as spark chambers. The signals produced in the counters when traversed by the particles are analysed by fast electronic logics: the arrival of a particle on the target in coincidence with the passage of a particle through the forward region and a particle in the backward region (at an angle between 70° and 160° with respect to the incoming beam direction) is recognized. Constraints are put on the velocities of the detected particles so that they are in the desired range; the decay properties of the particle travelling backwards are analysed in order to identify the $K^+$ mesons. When an event which satisfies the required conditions occurs, a signal (trigger) is generated by the fast electronics (with a delay of about 200 nsec) to activate the spark chambers.
The spark chambers (plus their read-out system) are devices employed to measure the coordinates of the transit point of a charged particle, localizing the position of a spark produced in the chamber when it is crossed by a particle. Placing more chambers along the particles' path, one is able to reconstruct the particles' trajectories in space. In our experiment (Fig. 2) two sets (telescopes) of four spark chambers each were set to cover, respectively, the forward and backward regions of interest; each chamber supplies two orthogonal coordinates (X and Y) of every spark, the total information being redundant to reconstruct the straight trajectories of the two outgoing particles. In addition, two more coordinates of a spark on the track of the backward travelling particle were given by a small cylindrical spark chamber, placed close to the polarized target in the region of a 25 kG magnetic field; this was intended to allow a magnetic analysis of the track, performed by correlating the position of a spark close to the target with the particle trajectory outside the magnetic field. The knowledge of the reacting particles' directions plus the momentum of one of the outgoing particles (the direction and momentum of the incoming particle being known from the average

![Diagram](image)

Fig. 2 Experimental set-up for the measurement of the polarization parameter in the reactions $\pi^+p^+ \rightarrow p\pi^+$, $\pi^+p^+ \rightarrow \Sigma^+K^+$ and $K^+p^+ \rightarrow pK^+$. In the picture, B1 to B4, R1 to R3, M0 to M4 represent scintillation counters. C1 to C4 and WATER C are Čerenkov counters. S1 to S4 and S5 to S8 represent the two spark chamber telescopes.
values of these two parameters in the beam) is sufficient to perform a kinematical reconstruction of the measured events; it is then possible to separate the good ones from the background leaking through the constraints set by the electronic logics\*). The final goal of the experiment is to calculate the distribution of the measured reactions according to the scattering angle, and to find the dependence of this distribution from the orientation (up or down) of the protons' spins in the polarized target. The final analysis is performed off line on a large computer: the small computer records the data and controls the proper running of the experiment.

3. **THE ON-LINE SYSTEM**\(^3\)

3.1 **Hardware configuration**

The on-line system has been implemented on a Hewlett-Packard 2116B computer, 16 k-word memory, 16 bit word, 1.6 µsec cycle time, two Direct Memory Access (DMA) channels. The computer is linked to the experiment through an electronic system\(^3,4\), designed and built in our group for the transfer of the spark chamber data, and a CAMAC interface, driving two CAMAC crates. A nine-track magnetic tape unit is used for the data storage; a teletype (with a recent addition of a 4010 Tektronix CRT display and a Logabax fast teletype) and a Tektronix 611 CRT display are employed for exchange of communication with the user. Two additional peripherals, paper tape reader and paper tape puncher, are used also for program development purposes. At present the computer, whose storage capacity has been increased to 24 k-words, is also equipped with a 5 M bytes disc unit and a FOCUS\(^5\) data-link to the CERN central computer centre: a system which will replace the existing one, in order to exploit fully the new facilities, is under preparation.

\* To give an idea of the background one should consider that, statistically, only one event over \(\sim 5 \times 10^5\) interactions in the target is of interest for this particular experiment.
3.2 Software configuration

The on-line software, written almost completely in ASSEMBLER language, is structured into the phases of initialization and data acquisition. In the initialization phase, entered when the program is loaded or when a new file is started on the tape, the user communicates with the computer through the teletype to specify the directives for the computer in the next run (number of events to be put in a file, histograms to build, histograms to print, etc.). The computer allocates the required space for the histograms, and sets all the necessary registers and parameters to their appropriate value, ready to receive data.

In the data acquisition phase a scheduler program coordinates, according to their priorities, the input of events from the experiment, event processing and transfer to tape, process control and exchange of communications with the user. An interrupt from the experiment triggers at any event the data acquisition routine. When an event is read in, the scheduler decides whether control has to be returned to the interrupted program (if the processing of a previously recorded event was still in progress), or if the completion of the interrupted operation has to be delayed. If, at the moment of the interruption, the computer was busy servicing a user request, endowed with a lower priority, the central processor is assigned to the event processing program.

When all the events are properly treated, the scheduler checks if a lower priority operation is waiting for completion or if a new service request has been received. In the affirmative case the operation is executed, otherwise the scheduler sits in an idle loop, waiting for the next event or the next request from the user.

The interleaving of operations is straightforward and does not require a stacking scheme (one has to keep track of not more than two suspended operations at a time). Nevertheless, conflicts can arise in the use of general utility (e.g. floating point algebra) routines, employed both by the event-processing and the user-request-servicing programs. The solutions usually adopted to overcome the problem -- that is, writing the routines in re-entrant code or appending a copy of the routine to each program -- were too space consuming. We decided to waste time rather than
Table 2

Core memory allocation (in K words)

<table>
<thead>
<tr>
<th>Task</th>
<th>Memory Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>0.5</td>
</tr>
<tr>
<td>Scheduler</td>
<td>0.5</td>
</tr>
<tr>
<td>Data acquisition</td>
<td>0.3</td>
</tr>
<tr>
<td>Data analysis</td>
<td>2.5</td>
</tr>
<tr>
<td>Data transfer</td>
<td>0.2</td>
</tr>
<tr>
<td>Process control</td>
<td>1.2</td>
</tr>
<tr>
<td>Communication with the user</td>
<td>3.6</td>
</tr>
<tr>
<td>General utility</td>
<td></td>
</tr>
<tr>
<td>(floating algebra, I/O drivers, etc.)</td>
<td>1.5</td>
</tr>
<tr>
<td>Storage and links</td>
<td>5.0</td>
</tr>
</tbody>
</table>

space. The lower priority program is divided into a number of stages: when a conflict occurs, the scheduler resumes execution, not at the interrupted point, but at the start of the appropriate stage. The storage allocations for the various activities is shown in Table 2.

3.3 Data acquisition

The trigger signal created by the fast electronic logics to activate the spark chambers, is also sent to the computer causing a high priority interrupt, to warn it that transfer of data is soon required. The computer assigns a DMA channel to the spark chamber data input, and sends a control word to confirm its ready status (the whole experiment is stopped if the control word is not received by the electronics). The relevant quantities to be recorded for each event are the positions of the sparks in the chambers with respect to a reference starting point for each chamber. The computer receives this information reading the contents of a single scaler, on which a quantity proportional to the spark position is digitized for each spark, one after the other. Owing to the characteristics of the
electronic system, which serializes the information from all the spark chambers through a cascade of magnetostrictive delay lines, the some 80 words to be read in arrive at the computer during a time of a few milliseconds. The digitizing electronics separates the information belonging to each spark chamber inserting in the appropriate positions within the string of spark coordinates a special code word. This will allow the computer to reconstruct the pattern of sparks in the later analysis.

At the end of a complete transfer from the spark chambers the computer assigns the DMA channel to the CAMAC interface, to read the contents of a crate containing scalers and pattern units. On completion of this transfer the system is ready for the next trigger.

The features of the data acquisition routine are fixed by the characteristics of the electronic system: as the coordinates to be read in are not permanently stored anywhere, the computer must be ready to respond to the event interrupt with minimum delay. The coordinates' arrival at the computer is spread at random over a time of some milliseconds: the use of DMA makes the input operation more efficient.

The events rate is limited by the spark chambers, which need a recovery time of 10 to 20 msec after each trigger. The beam hits the target during bursts of \( \sim \) 500 msec, followed by a beamless inter-burst period of \( \sim \) 2.3 sec. These factors combine to produce a maximum event input rate of 6-7 events/sec, leaving the computer enough time available for other operations.

3.4 Data transfer

The storage space reserved for the data from the experiment is virtually divided into \( n_1 \) "records", each one composed of \( n_2 \) "events". Each event is formed by \( n_3 \) words for the spark chamber data and \( n_4 \) words for the CAMAC scalers. The four quantities \( n_1, n_2, n_3, \) and \( n_4 \) can easily be redefined in the program, giving the possibility of changing the storage configuration to take into account variations in the trigger rate, the expected average number of sparks per event, the number of scalers to be read, and so on. As the events arrive, they are successively stored in the "event" cases, filling the space in a circular way. When an event should be written in an unemptied case, the data acquisition is stopped and a warning is sent to the users. The events are processed as soon as
they come in: the scaler information is partially decoded, and the result is overwritten on the raw data memory locations. Whenever $n_2$ events are processed, a record is transferred to the magnetic tape. Normal figures for the mentioned quantities are:

$$
\begin{align*}
    n_1 & = 3 \\
    n_2 & = 10 \\
    n_3 & = 80 \\
    n_4 & = 16 .
\end{align*}
$$

A transfer of data to the central computer through the data link is at present performed off-line, reading the contents of a file pre-written on the magnetic tape. It is envisaged to perform such a task on-line, in parallel with the recording on tape.

3.5 Data analysis

The measurement of the angular distribution of the particles involved in the reactions under investigation, obtained from the information provided by the spark chambers, is the final aim of the experiment: a prompt reconstruction of the particles' trajectories through the spark chambers is of fundamental importance both for monitoring the physical content of the data collected and for checking the correct operation of the spark chambers, the most critical part of the experimental apparatus. The track reconstruction is performed at a first stage in our experiment by the on-line computer, which correlates the measured positions of the sparks in the chambers, searching for four sparks or three sparks alignments (two sparks alignments are disregarded in order to limit the background of spurious reconstruction and to relieve the work of the on-line software). The track-finding routine, the same FORTRAN routine employed in the off-line analysis programs, reconstructs, for each of the two spark chamber telescopes, the projections of the tracks on the XZ (horizontal) and YZ (vertical) planes, Z being the direction normal to the telescope. The outputs from the routine are the $m$ and $q$ parameters of the line obtained by joining the first and the last spark belonging to the reconstructed track, plus a flag identifying the non-contributing chamber in the case of only three sparks alignments.
Fig. 3 Picture of an event reconstructed on-line. The upper part of the picture represents a view of the experimental apparatus projected on the horizontal plane. The beam enters from the right. The recorded sparks and tracks reconstructed in the two spark chamber telescopes are shown. In the lower part of the picture, the same two telescopes are shown in a side view. The figure in the right-hand bottom corner of the picture represents a set of scintillation counter strips: the mark shows which counter was hit.

Fig. 4 Spark chamber efficiency table. The numbers under the headings MESON, BARYON and ROUND AND T'S are the percentages of spark multiplicities for each chamber (no spark, one spark, two sparks, more than two sparks). The percentages of track multiplicities (no track, one track, two tracks, more than two tracks) appear for each of the four coordinates planes under the heading MX MY BX BY. The percentage inefficiency of each spark chamber plane and the track multiplicities over a whole telescope and over the whole apparatus are then displayed.
Fig. 5 Distribution of the horizontal coordinates of the intersections of the tracks with a vertical plane through the beam axis, measured along the same line. Owing to the magnetic field crossed by a particle coming from the target, a positive coordinate for the projection is associated with a positive charge and a negative coordinate with a negative charge.

The information obtained from the track reconstruction is employed to display on the CRT screen, at a frequency chosen by the user, the pictures of the reconstructed events (Fig. 3). In addition a table is set up to give the percentage of track multiplicity in each projection plane and the inefficiency of each spark chamber (Fig. 4). Moreover the program can calculate distributions of the intersections of the reconstructed tracks with any chosen plane parallel to each telescope. Such a facility has had several applications, like building beam profiles in the setting-up stage of the experiment or, during the data-taking runs, for instance to monitor the sign of the outgoing particles' charge (Fig. 5).

The data-processing routines also perform a kinematical analysis of the reconstructed events in order to provide on-line distributions of the coplanarity angle and the kinematical angular correlation (see Fig. 6) for the elastic reactions $\pi^+ p \to p\pi^+$, $K^+ p \to pK^+$; for the inelastic events of the type $\pi^+ p \to \Lambda^+ K^+$, missing-mass spectra are calculated by joining the information on the outgoing $K^+$ meson direction with the measure of
Fig. 6 Kinematical angular correlation for the reaction $\pi^+ p \rightarrow p \pi^+$. The histogram shows the difference between the measured $\pi^+$ scattering angle and the same angle as computed from the proton recoil angle assuming the $\pi^+p$ elastic kinematics. The peak corresponds to the elastic events.

its momentum obtained through the cylindrical chamber near the target*).

Owing to the low speed of the floating-point arithmetics, such a form of elaborate analysis has turned out not to be efficiently performed by the small computer and could be executed only on a sample of the collected events: rewriting the track-finding routine in ASSEMBLER language and double precision fixed point arithmetics and committing parts of the analysis task to a bigger computer, fed through the data-link, will permit a more efficient utilisation of the small computer.

3.6 Process control

Of all the events taking place when the beam hits the target, only a few are completely measured by activating the spark chambers: the selection is performed by the electronic logics which analyses the signals from the scintillation and Čerenkov counters, in searching for the right coincidences. Both the counters and the electronic logics are to be kept

*) A list of all the histograms built by the analysis routines is given in Table 3.
Table 3

Histograms available on line

Contents of scalers and pattern units.
Total number of sparks per event.
Density of sparks projected on any axis of a spark chamber.
Coplanarity angle for the reactions $\pi^+ p^+ \rightarrow p\pi^+$ and $K^+ p^+ \rightarrow pK^+$.
Kinematical angular correlation for the same reactions.
Meson time-of-flight for the reactions $\pi^+ p^+ \rightarrow p\pi^+$, $K^+ p^+ \rightarrow pK^+$, $\pi^+ p^+ \rightarrow \Sigma^+ K^+$.
Momentum of the scattered meson for the $\pi^+ p^+ \rightarrow \Sigma^+ K^+$ reaction.
Mass of the scattered meson for the $\pi^+ p \rightarrow \Sigma^+ K^+$ reaction.
Missing mass of the recoil baryon for the $\pi^* p^+ \rightarrow \Sigma^+ K^+$ reaction.
Intersection of reconstructed tracks with a chosen plane parallel to the spark chambers.

under control to monitor the stability of their performance: these checks are periodically performed by the computer through a series of CAMAC operations. All the counters (about 40) are connected through a multiplexer to a single scaler: on request, the computer opens in sequence each multiplexer channel, reads the counts over one second and produces a table, to be compared with a reference one. Another set of scalers is employed to record the counting rates at various levels along the electronic logics chain: on user request the scaler contents are read, normalized and displayed in another table to be checked for stability. More control operation should be performed, for which the appropriate hardware must be designed: a CAMAC analogue-to-digital converter plus a multiplexer are now ready to read the voltages supplied to the counters.
Table 4

User requests recognized by the computer

<table>
<thead>
<tr>
<th>Request Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display the events at a desired frequency.</td>
</tr>
<tr>
<td>Display the efficiency table.</td>
</tr>
<tr>
<td>Reset the efficiency table.</td>
</tr>
<tr>
<td>Display a histogram.</td>
</tr>
<tr>
<td>Change the origin and the binning of a histogram.</td>
</tr>
<tr>
<td>Display the spark coordinates and the scalers contents for the last event.</td>
</tr>
<tr>
<td>Read the counting rates of all the counters and display the table.</td>
</tr>
<tr>
<td>Read the counting rates along the electronic logics chains and display the table.</td>
</tr>
<tr>
<td>Dump a selected area of memory.</td>
</tr>
<tr>
<td>End the file.</td>
</tr>
</tbody>
</table>

The currents in the beam transport magnets, the gas flow in the spark chambers, the gas pressure in the Čerenkov counters and the target polarizing apparatus are more things the computer could control. As a next step the computer should be able to vary the instrumental parameters (voltages, delays, etc.), giving the experimenters the possibility of optimizing the experiment just sitting at a (remote?) console.

3.7 Communications with the user

The computer’s supervision work is made available to the user through three different types of operations:

1. Fulfilment of a request received from the keyboard through a special code: the requests recognized are listed in Table 4.

2. Display of the reconstructed events (Fig. 3) at a frequency chosen by the user.

3. Print-out of tables and histograms produced during the previous run.
Operations (1) and (2), which follow the staging rule, can be suspended to process the events but cannot suspend each other. Operation (3), for which no staging is required, can be suspended at any moment to process the events and can also be suspended by a request of types (1) and (2) at the end of the printing of a complete line. The scheme is suitable for expansion, with the addition of more background operations.

REFERENCES

1) CAMAC - A modular instrumentation system for data handling, EURATOM 4100e (1969).
CAMAC - Organization of multirate systems; EURATOM 4600e (1970).


3) See also:

4) A. Vascotto, A digitizing system for magnetostrictive wire spark chambers. (In preparation for Nuclear Instrum. Methods.)

5) D. Ball, P.M. Blackall, V. Gerard, G.R. MacLeod, P.J. Marcer and E.M. Palandri. FOCUS - A remote access file handling system on-line to a CDC 6000 series computer, Comput. J. 14, 113 (1971).
ACCELERATOR CONTROL

R. Keyser
CERN, Geneva, Switzerland

1. INTRODUCTION

This talk considers the role of a small computer system in controlling the ISR. The present aims of the system, and the characteristics of the tasks it carries out are described. The necessary facilities to carry out these tasks together with those parts of the system organization particular to a control environment are considered.

2. IN PERSPECTIVE

2.1 The ISR

The Intersecting Storage Rings (Figure 1) are two slightly deformed toroids into which are injected bunches of protons that have already been accelerated to high energies by the Proton Synchrotron. These bunches are stacked by radio frequency acceleration until currents of several amps are attained and then kept circulating for several hours. The protons circulate in opposite directions in the two rings allowing almost head-on collisions in eight intersection regions. The toroids in which the beams circulate are evacuated to give an ultra-high vacuum with pressures in the region of $10^{-10}$ torr.

There is a large and varied range of instrumentation which monitors and controls the process and which is connected to the control computer.

An accelerator is usually unique and so is a piece of experimental apparatus in its own right. Thus it is not only designed, built and made to work but is continually evolving. New instrumentation is added and old instrumentation improved. These changes have a strong effect on the software and hardware requirements.
2.2 The Computer System

The computer system has the configuration shown in Figure 2. Further details may be found in (1) and (2).

Briefly, a few remarks concerning computer 2. This is used for the development of hardware and software. The ISR itself is being used about 70% of the time but the vacuum system is maintained 100% of the time. Thus a separate facility is needed to develop new hardware and software. Computer 2 has reduced peripherals but an almost identical operating system.

A multi-programming system has been written which contains input/output drivers, a core store scheduler and a disc scheduler queues requests for the execution of programs and controls the running of up to seven application programs which may be in core at a given instance. In practice one of these levels is occupied by the command interpreter which receives the commands typed in by operators, a further level is used for core resident programs which need to do frequent scans (order of one per sec) and a third level (the lowest) is given over to alarm monitoring and allied work.

The disc is used to hold programs and data. Programs are held in a link-edited form or disc but with relocation bits. They are brought down into store for execution and relocated each time they are loaded.

A simple disc scheduler is implemented with the following characteristics:

(i) Files are referred to by four-character names with a two-character extension. Operators can create, delete and write into data files by specifying the four character name as a parameter in their command. The program itself adds the two character extension. An operator can now create a file FRED say for the vacuum system and can rest assured that even if another operator creates a file FRED for some other application, the computer system will use the correct FRED in its context.
(ii) A file occupies a contiguous area of disc. Not only does this simplify the implementation but it greatly reduces the time taken to transfer a file to store. The dataway between the disc and core is a potential bottleneck in any computer system.

(iii) Files can be given a software status of "permanent" thus preventing accidental overwriting by an operator.

(iv) Once a file is created it cannot be extended though it can be reduced in length.

(v) A simple index in core gives the block (or sector) on disc where the directory entry of the file is to be found. For the first read or write to the file (at least) two disc accesses are required. Thereafter director information is kept in core until the program closes the file or the program terminates. Hence subsequent file reads or writes may require only one disc access.

2.3 User Software

Until the beginning of this year all programming including that for application programs was written in assembly language. No suitable alternative was available. Early this year a CORAL 66 compiler became available and was integrated into the system to work under the machine organiser. Experience with CORAL is very satisfactory and the following comments are appropriate.

(i) the average increase in program size due to writing in this high level language is of the order of 20%.

(ii) the language and the computer are well matched. Virtually all the instructions in the computers' repertoire are used. In particular since the ARGUS is a word machine of 24 bits, the bit handling and part word facilities in the language permit the programmer with no extra effort to efficiently use core storage. Where in assembly language the programmer, for speed, ease and accuracy of writing would use a small field in each word, with CORAL he packs a number of such fields into one word.
(iii) it is easy to break into machine code where it is necessary. Such flexibility is essential in a process control environment where the program has to work in real time and the programmer is intimately involved with input/output processes.

It was realised early on that a run time debugging package was required. This is now an integral part of the system software and allows the running program to be stopped at chosen places and the contents of core and registers examined and altered. Further the disc copy of the program may be modified. Such a package is particularly relevant to small machines where the programmer is in immediate contact with the computer.

The debugging package proved a useful tool for use with a program written in assembly language. While it can be used with a program written in high level language, one needs to work using an assembler listing of the program. One can pose the question, is it not possible to incorporate the debug facilities in the implementation of the high level language? The answer is 'yes'; one solution is to use an interactive interpreter.

An interpreter works by interpreting each high level statement each time it is has to be executed rather than compiling code. The routine that interprets a particular statement need only be brought into core when it is needed, overlaying the routine used for the previous statement. Thus a program run interpretively may use only a small fraction of the core that would be needed if it were compiled though at the cost of speed. Thus an interpreter is particularly appropriate for a small machine in that it provides a suitable structure on which to implement a run-time debug facility and also that it uses core efficiently which is a much more critical commodity than central processor time.

3. **THE TASKS TO BE EXECUTED**

Let us now review the various types of tasks that the control computer is expected to provide. Detailed description of various applications can be found in (6) and (7).
3.1 Monitoring

Monitoring is the periodic scanning of digital data for faults and the output of a message to the operator in the event of a fault being detected. Similarly by periodic scanning of analog data, drifts in apparatus can be detected and the operator suitably alerted.

Monitoring also includes surveillance of other asynchronous events such as the deliberate manual intervention of an operator on some equipment. The requirements that are placed on the computer system by such asynchronous events are similar to those of faults so to simplify the discussion faults will be taken for illustration.

It should be clearly understood as to what a computer can and should be asked to do with regard to monitoring. Firstly there is the long time taken by the computer system to recognise and act on a signal, compared with special purpose hardware. Thus actions that must be executed immediately for the security of the machine or personnel must be implemented by special purpose hardware.

The monitoring of a status many times a second requires a core resident program - usually imbedded or tightly linked with the machine organiser. It is not simply a matter of writing an application program.

A monitoring job on detecting an abnormality alerts the operator by means of a message. That this takes time is of little consequence since the operator will have to correct the fault and that often takes considerable time. Thus the time scale in which the computer works for monitoring jobs is in terms of minutes and so the frequency of the computer scan can be in terms of minutes.

Whilst this is true for many monitoring jobs, it is not necessarily true for all of them. For example, the required response time of the computer to an operator pushing a switch is more likely to be required to be of the order of hundreds of milliseconds. Ways must then be sought for reducing the total load on the computer. One such way is to have an extra signal which is set when any one of a number of faults or other events occurs. The computer then has to check only one status
bit and only when the event has occurred, does it have to carry out a full scan. This can be further extended so that a change in one of a number of status bits causes an interrupt. The scanning program need only run when a fault is known to have occurred, and moreover, it will give a quicker response to the fault.

3.2 Operation Programs

It is appropriate to consider the computer system as a powerful tool for use by the operator. During routine running, that is preparation of beams for physics and during physics runs, many of the tasks of the operator are well defined. As far as they are well defined a program can be written to perform the task. The operator needs easy and convenient means by which he can communicate with the computer to initiate complete programs and a clear response from the computer. We will return to the man-machine interface shortly.

We are not yet at the stage when the operator can press a button and the computer controls the instrumentation in such a way that a stacked beam ready for physics is produced. (There are dangers in this - real control passes out of human hands into a machine in many ways). At present various parts of this process are automated. Other parts are not automated due to the qualitative nature of the decisions to be made. The computer can only help the operator make his decision by providing the relevant data in a clear form. If we take a look at some of the tasks carried out by the computer the advantages and general philosophy of computer control will become apparent.

(i) Setting of magnet power supplies

There are some 100 power supplies that have to be set in the beam transfer system and another 200 in the rings themselves. Not only is the computer considerably faster than a human operator, it is meticulous in following the given procedure and carrying out the checks.

Moreover, there are certain tasks involving the power supplies of the ring magnets which could not be carried out by an operator without a special facility. Now the power supplies are already connected to the
computer so if the special facility can be provided by a computer program, this solution is much cheaper and more flexible than providing a piece of special hardware. An example of such a task is to change the currents in some twenty-four or more power supplies all at the same time and in a constant ratio.

The computer is well able to exploit the basic technique used to set up the ISR. This consists of recording the values of controllable parameters, notably power supply currents, at the end of a successful set up involving individual adjustments to certain parameters. The next time the ISR is required to be set up in a similar manner the computer extracts the values from its file store and sets the instrumentation to these successful values. Unfortunately repeatability in other parts of the complex is not attainable but at least the set values are a good approximation from which to start adjustments. This is exactly the procedure which is used for the beam transfer, a final adjustment to the last two magnets being made by a beam optimisation program which adjusts these two currents until the excursion of the injected bunches is a minimum.

(ii) Beam observation

In each ring there are 54 beam observation stations, each station giving four signals. These signals are read via the analog scanner, corrected for equipment offset, scaled and then manipulated to give the position of the beam (in this case a single pulse from the PS).

Each signal has its own offset and scaling factor which drifts slowly but remains valid for several hours. On command from the operator the computer generates the test signals from which the offset and scaling factors can be calculated and stored on disc. Since the means for generating these test signals have to be provided in any case, it is little extra effort to allow the computer to generate them especially if the function is put in at the design stage.

To take a beam orbit measurement requires synchronisation between the computer and the external process. The computer program must send
out a trigger to activate the beam observation station. This is delayed (gated) by hardware until a further trigger which is synchronised with the beam arrives. The beam observation stations then react to the passage of the beam producing a steady voltage which can be sampled by the analog scanner. After a suitable delay of the order of a few milliseconds to allow settling of the electronics the computer will receive an interrupt which reacts the beam observation program.

The program reads the steady voltages over eight channels of the analog scanner which takes almost ¼ second. Since the steady voltage is not completely steady and decays significantly over periods of a minute and further there may be another pulse in 2 seconds whose position should be measured, the program must run with high priority. Both the central processor and the analog scanner will stop any work they were doing for other application programs and execute the work for the beam observation program. The central processor and the analog scanner are the only two cases where the software allocates resources on a priority basis.

The beam orbit is used in two ways and the manner of presentation must be adapted to the operator's needs. Firstly the operator may wish to see it immediately for qualitative evaluations and for this it would be displayed on a CRT screen. Secondly, he may wish to keep a record of beam orbits over periods of weeks and for this a 'hardcopy' is required. This gives a printed form of the picture similar to that displayed on the CRT screen but also including the values of the displayed points.

(iii) Vacuum

The vacuum system has over 700 pumps and gauges together with valves and other equipment. Much of the equipment needs to be automatically calibrated and the readings corrected. Its very size accentuates certain problems rather than raising new ones.

In order to overcome the size of the problem the technique used is to allow the operator to specify a subset of interest. For example he can easily see a pressure bump on the CRT display of the
pressures around the whole ring. This picture however takes eight seconds to produce. If he wishes to follow the evolution of this pressure bump then he must be provided with the means of selecting only those gauges in the area of interest, in order to increase the scanning rate.

3.3 Development

The computer system has to provide facilities for its own development and for the development of the accelerator itself. Facilities above those noted in the previous section need to be given to developers who are of two distinct groups. They are the instrumentation developers who work on a very elementary level and the accelerator developers. The first want to know if a certain bit is being correctly set in the digital scanner or if the computer is recording a certain signal correctly. The second group of developers want routines which drive individual bits of equipment and want to be able to link these routines together to form a program. These two groups need to interact with the system. One way is to provide a special program for their individual needs. An alternative and more general solution is to provide an interactive interpreter and a number of basic routines and let them link the routines together to suit their purpose.

4. COMMUNICATIONS

A discussion on man-machine communication techniques is outside the scope of this talk. It is an area where attention to small details is required and where consistency, clarity and simplicity are vital.

At the ISR the operator's primary means of issuing an instruction to the computer is via a keyboard. Any input on a keyboard that is not reserved by an active program is directed to a core resident command interpreter which accepts commands of the general form

\[
\text{label : program name, clauses (parameter list)}
\]

Clauses are decoded by the command interpreter. They allow the operator to specify to the program on which device it should obtain its input or
output its results. Further clauses are used to activate a program a number of times at specified intervals or in synchronisation with the PS cycle.

The parameter list is passed on to the program and is interpreted according to the conventions of the program. It is necessary to keep the parameter list in core store until the command with all its repetitions is exhausted.

Let us take a hypothetical example:

INCM, CR, IN = 1P(2)99 (1H816)

If the program INCM incremented a specified magnet, in this case the magnet numbered 1H816, by an amount specified on an input device, then the program would read this amount from a card (CR) and would run on the next PS pulse and then every-second pulse for a total of 50 executions.

These commands give the operator considerable flexibility. If things go wrong he may have to use this flexibility but for operational purposes a well-defined subset is used. In order to avoid the necessity of typing in often-used commands each time with its attendant risk of error he may attach a command to a button and issue the command by pressing the button. To do this he uses a utility program that forms a file on disc containing up to eight commands. He may then at any time attach this file to any one of four sets of eight buttons.

5. CAMAC

The principles of CAMAC have already been covered in an earlier talk (3). It has two particular relevances to the ISR control computer. Firstly due to the wide range of instrumentation of the ISR a wide variety of interfaces are necessary for the computer. CAMAC specifies common input and output rules that these interfaces should obey greatly facilitating the problems that the various engineers and physicists face in connecting their equipment to the computer. Further, there are many CAMAC modules for the computer commercially available which can be directly plugged into a CAMAC crate and are then
immediately accessible by the computer. In such cases connection of equipment to a computer is a system engineering problem and design and development effort is avoided.

Secondly, there are a number of computers controlling experiments around the ISR. The ISR control computer has access to variables which the experimenters require (such as beam current) and the experimenters' computers have variables which the ISR control computer would like in order to provide better operating conditions (such as background and event rate). Clearly a data link between the computers was required and since CAMAC was common to all computers it was logical to develop a CAMAC module, standard for all types of computers, which when plugged into their CAMAC crates attached them to the data network. This has been implemented and is operational for some of the experimenters' computers.

There are some problems in the implementation of a CAMAC driver in a multiprogramming system. The first is how can one realistically maintain system integrity. By this is meant how can the CAMAC routines in the machine organiser ensure that two application programs can concurrently use the CAMAC highway without interfering with each other. One can envisage a system similar to the classical system for magnetic tapes, whereby an application program executes a coded jump to organiser and the organiser tests this for legality and translates the jump into function code, module and subaddress. But this is expensive in terms of time and especially core, view the variety of equipment that has to be driven. To avoid these overheads it is necessary to throw back some of the responsibility onto the application programmer.

There are two categories of interference possible during the attempted concurrent use of CAMAC equipment. The first category is programming errors and the second is attempted legitimate use of a CAMAC module before another program has finished using the module. The responsibility for avoiding errors is placed on the application programmer. The machine organiser retains the responsibility of avoiding the second category of interference. It achieves this by the method of semaphores whereby an application program has to request the use of a
subsystem of the CAMAC and it will be suspended if this subsystem is already secured by another application program.

Trouble from the first category can be virtually avoided by insisting that an application program accesses a CAMAC subsystem only through library procedures or standard macros. The procedure approach is favoured since it is more general and better suited to high level languages. The procedure can be considered as an extension of the machine organiser and should cause no more debugging problems than a routine imbedded in the organiser to carry out full checks. This approach does have one minor drawback in that repositioning of the modules in the crates cannot be carried out without reassembling of programs.

6. THE SOFTWARE IMPLEMENTATION

We have looked at some of the characteristic tasks that are performed by the ISR control computer. Let us now look inwards to see how a computer is organised to provide the necessary facilities. This has been described in general terms in the lectures of I. Pyle (5). Here we wish to consider some of the more important aspects with regard to the ISR implementation and for this discussion we refer to hierarchy shown in Figure 3.

6.1 Hardware interrupts

A process control computer has attached a wide variety of equipment, each piece usually associated with one or a number of interrupts. Further interrupts are required for synchronisation with the external process. Interrupts influence the software in a process control environment to a larger extent than in other computer applications. Even the operator has to be aware of interrupts, for example his ability to trigger a program on receipt of a specified interrupt, or to set an external trigger pulse for the program to take a reading of some instrument at a given time in the accelerator cycle.

The response of the computer to a hardware interrupt is twofold. A simple response such as read and store a word from a piece of equipment
would be done immediately without further effects. If the response is more complicated then flags will be set to carry out further tasks after processing other hardware interrupts. This structure ensures that any interrupt has a chance of having a simple response given within a short time of about 20 msec. This is sufficient for our environment, though a computer with priority interrupt levels could substantially reduce this time. However, the main advantage of interrupt priority levels is usually in the reduction in overheads incurred in changing from one interrupt routine to another. If a piece of hardware equipment needs a fast response it is usually due to inadequacy in its design.

6.2 Software interrupts

Software interrupts have three sources. Firstly the flags produced by the hardware interrupt routines, secondly by routines that handle the software interrupt and thirdly by requests from programs on lower levels, such as an application program.

A software interrupt is serviced by one or more routines. Such routines are the peripheral handling routines, the core store scheduler, the termination routine, etc. One of particular interest is that which activates application programs. It is also an example of why the three sources of software interrupt are treated similarly since programs can be activated either on a hardware interrupt, on termination of another application program when core becomes available, i.e. via the termination routine or thirdly by a request from an application program.

6.3 Program Activator

In general programs are kept on disc and brought down into core when a suitably large contiguous area of core is free together with the availability of a multiprogramming level.

The program activator has a table of tasks to be performed, consisting of control blocks giving the necessary details such as name and size of program. These control blocks form a number of linked queues with backwards and forwards pointers. Each interrupt that can initiate programs will have its own queue together with a queue for
programs initiated on time and a further queue for programs that will be run as soon as core and multiprogramming level are free. When an interrupt occurs that might activate a program the associated queue is scanned to see if it is appropriate to activate a task. (A task may not be executed each time the interrupt occurs but every nth time). If it is to be activated then it is temporarily removed from its queue and added to the core and level queue. When the task is terminated either its control block will be deleted (single activation) or returned to its original queue (multiple activation). Having processed an interrupt queue or the time queue (in a similar manner) the program activator will search for core and a level. If it finds both it will reserve them and load into that core a bootstrap routine which loads and reallocates the task from disc. The bootstrap routine will be supplied with necessary information from the control block.

6.4 Core store allocation

The static core map is shown in Figure 4. A number of frequently used routines such as floating point arithmetic, input/output etc. are written re-entrantly and stored in core. Thus one copy in core suffices for all the active application programs plus the core resident programs. Not only do the re-entrant routines save core store but disc store also since multiple copies of these routines need not be stored on disc.

There are certain programs which are core resident since they are called frequently to run. These include the command interpreter and programs which scan contacts every one second. The former needs to be core resident otherwise unrecoverable situations can occur, e.g. if a deadly embrace occurs the operator must be able to abort one program. An example of a program which needs to run every second is that which scans the program request buttons to determine if the operator has issued a command by that means. To call it every second from disc would place an unacceptable (25%) loading on the disc.

Finally 12 K of core is left for application programs. The core store scheduler allocates a program just enough store for it to be
loaded into plus 256 words for the relocation routine which can later be used by the program as work space. Thus programs run in variable partitions, positioned in store according to where there is a sufficiently big free space.

REFERENCES

(1) R. Keyser: The dual ARGUS 500 Control Computer System for the ISR - ISR-CO/69-45

(2) P. Wolstenholme: Data transmission and computer application for the CERN-ISR Project - CERN/ISR-CO/70-42

(3) H. Verelst: The ISR Data Network - CERN/ISR-CO/71-59

(4) B. Zacharov: System aspects of small computers in elementary particle physics - Proceedings of this School.

(5) I. Pyle: Small computers in physics - Proceedings of this School.


Fig. 2  The ISR Argus 500 twin computer control system
Fig. 3 The software organisation in the ISR Argus
Fig. 4  The core store map in the ISR Argus
SIGNIFICANT CONCEPTS IN DATA BASE MANAGEMENT

T.W. Olle
Independent Consultant, Branksome, Poole, Dorset, UK

1. INTRODUCTION

The purpose of this paper is to tie together the material presented in a series of nine lectures at the 1972 CERN Computing and Data Processing School. The presentations were based on two reports prepared by CODASYL committees and both released during 1971\textsuperscript{1,2}). The present paper attempts to distill the highlights without unnecessary duplication of detail.

2. LOGICAL DATA STRUCTURES

This treatment follows that presented in Chapter 2 of the CODASYL Systems Committee report\textsuperscript{1}). Logical data structures are identified as the user's view of the data, without regard for any mapping which it may undergo prior to storage in physical media. The aim of the committee's analysis was to build a framework of features in which a selected number of recently developed (and in some cases also commercially available) systems could be represented. This is important in the sense that the report therefore does not attempt to provide a complete all-embracing theory of logical data structure.

In the concept of a data structure class for a given system, the system designer must select from six structural elements which are identified as

- Item
- Group
- Group relation
- Entry
- File
- Data Base.

A simple system might choose only three, for example, item, entry, and file. It should be noted that the term "entry" is perhaps better thought of as a "record". Although the analysis was oriented to ten existing
systems, the techniques and concepts should be valuable to any designer undertaking the definition of a generalized facility for processing data.

Much of the complexity introduced in the report stems from the frequent requirement in commercial applications to represent the data in either hierarchical or network structures. This requirement is stimulated by the desire to integrate the data files from previously segmented application into a centralized data base to be used by several applications, possibly concurrently.

3. DATA DEFINITION

The process used by a given system to define data, as opposed to its facilities for representing such data, is analysed separately in Chapter 3 of Ref. 1. A brief survey of the form of the data definition language is followed by an examination of the important concept of a given system having two data definition languages. One is for use by a data administrator who is regarded as having both initial and on-going responsibility for the overall structure of the data base. The other is for use by an individual programmer in defining the view of the data (essentially a sub-structure) for use in one or more programs to process the data.

4. SELF-CONTAINED INTERROGATION FACILITIES

Generalized data base management systems are essentially regarded as being in one of two principal classes, called host language and self-contained. The former class is for use by a programmer and essentially provides an extension to some existing procedural programming language, such as PL/1 or COBOL. A self-contained system provides its own language and is regarded as being an easy-to-use system for a non-programmer.

Probably the most important facility of a self-contained system in any user environment is that used for interrogating files and generating a report containing data retrieved from the file or files. Such a facility is sometimes referred to as a query language (or even a report generator), but in Chapter 4 of Ref. 1 is identified as an interrogation function.

An interrogation function consists of up to four parts — selection (conditional expression), extraction, processing, and formatting. An important difference among systems is the range of applicability of the
function to the data structure. Some are restricted to interrogating a single file of similarly structured entries. Accepting this most simple situation for the purpose of further illustration, the selection part of the function is always required and has the role of selecting certain entries whose content satisfies a potentially complex conditional expression.

Extraction is the process of picking out from each entry certain items for inclusion in the output from the function. Processing implies either sorting the extracted entries according to a sort key, which is usually one of the extracted items, or else performing some computation on some of the numeric data items which are extracted. A modest interrogation function might in fact not provide any kind of processing facilities as here defined.

Finally, formatting is the process of arranging the extracted data (possibly after processing) into either a printed report for human perusal, or else in a computerized file for subsequent further processing. The range of formatting capabilities varies widely.

5. SELF-CONTAINED UPDATE FACILITIES

Update (see Ref. 1, Chapter 5) is a function somewhat analogous to interrogation, but with basically two parts -- selection and action. Selection may be performed in a similar way to interrogation, namely by evaluation of a potentially complex conditional expression on the entries in a file. More frequently it is the direct selection of a specific entry based on the value of one of its items, such as an employee number or an experiment number.

The action can usually be the insertion, deletion, or modification of an entry. If modification is to be performed, this implies a change to the content of an entry, normally the modification of the value of some item in the entry.

Interesting variations in the capabilities of update functions come from the approaches to handling a multiplicity of transactions and also from the ways in which these can be validated.
6. **PROGRAMMER FUNCTIONS**

A host language system provides programming user facilities (see Ref. 1, Chapter 7) to manipulate a data base. The facilities are on a different level from those in a self-contained system. In a host language system, the programmer's statements are mostly on the entry or group level. He is able to "locate" an instance of a group, or in some cases "locate and access" such an instance. The difference between these two is that after a "locate and access" he can then perform item level operations on the data contained in the group, using the facilities provided by the host language (COBOL, PL/1, etc.). After an execution of a "locate" statement, a corresponding "access" must be executed on the same group instance before item level host language statements can process the data. It should be noted that systems providing network structure prefer the two separate statements; those restricted to hierarchical structures provide the combined statement.

Statements for use by the programmer are classed as

- control (e.g. OPEN)
- retrieval (e.g. LOCATE)
- modification (e.g. ADD).

In addition, a system may adopt its own approach to "housekeeping" functions. The analysis in Ref. 1 discusses among other features

- invocation from host language
- addressable structures
- currency
- user work area
- security clearance.

7. **STORAGE STRUCTURE**

The topic of storage structure is one on which numerous books have been written\(^2\),\(^3\)\(^)\). To many the techniques for structuring data in storage are the epitome of data base management concept. However, in Ref. 1, analysis of storage structure was postponed until Chapter 8 because, in the data base management systems (dbms) analysed, the storage structures are essentially "behind the scenes". An approach is chosen by the dbms designer and the user may in certain cases have no further choice to make.
"User control over storage structure" is in fact one of the more important features analysed.

The chief interest in storage structure is on the file level, namely how entries and groups are stored in a file. There is most flexibility when the storage media type used is direct. Approaches may be classed broadly as pointer, hashing, and indexing, and each of these has a wide variety of sub-classes.

8. **DBTG PROPOSAL**

Of major importance for anyone seeking insight into the current state of the art in data base management is the proposal submitted by the CODASYL Data Base Task Group in April 1971 to its parent committee.

This proposal contains detailed specifications for the following three components:

- Schema data description language (DDL)
- Sub-schema data description language for COBOL
- COBOL data manipulation language (DML).

The first component is regarded as being independent of any processing language. The second two are specified for COBOL, but it is emphasized that the DBTG would like other groups to use these as prototypes to specify corresponding facilities for other processing languages such as FORTRAN.

The most important structural element in the DBTG proposal is that of a "set" essentially the same as that which the Systems Committee refers to as a "group relation". The set is in fact "a named relationship between two or more record types". One record type must always be declared as the set owner, the other (or others) as members. Each set type may occur several times in the data base. In fact, it may help to equate an "occurrence of a set" to the more widely understood concept of a file. The chief difference is that the set occurrence must have an owner record occurrence and may have many occurrences of each member record type.

9. **CONCLUSION**

The concepts embraced by the term "data base management" are increasingly numerous. An attempt has been made in this short paper to point out
the most important ones as well as indicating potential ramifications. The reader is referred to the main references\(^1,2\) for full details.

\*

**REFERENCES**

1) CODASYL Systems Committee, Feature analysis of generalized data base management systems, May 1971, 520 pp. (Available at $8.50 pre-paid from IAG HQ, Stadhouderskade 6, Amsterdam 1013, Netherlands, or from British Computer Society, 29 Portland Place, London WIN 4AP.)

2) CODASYL Data Base Task Group, April 1971 report. (Available at $6.50 from IAG and BCS -- see Ref. 1 for addresses.)


INTRODUCTION AND SUMMARY

Under compilation we understand the translation of high level language programs into machine code. The more often high level languages are used for programming the more important is correct and efficient translation of programs. Correct translation can only be achieved if syntax and semantic of programming languages are clearly defined and strictly obeyed to by compiler constructors.

In these lectures we will present a simple extendable scheme how syntax and semantic can be defined rigorously. This scheme fits to many programming languages, especially to ALGOL-like ones. We will consider statements and programs to be notations of state transformations, in special cases storage state transformations.

Assignment statements are used to show what syntax and what semantic are together with them, and what kind of considerations are necessary to create correct and efficient translation methods. Because of their practical importance, arrays and subscripted variables are discussed intensively. Especially automatic recursive address calculation methods can decrease computing time of programs quite considerably.

Our lectures do not cover all that could be incorporated in theory and practice of compilation. We will mention procedures and their implementations, formal languages and their analyzing methods. It is not the aim of the lectures to give a very wide-spread survey but to show that compilation can be made transparent not only to computer scientists but also to physicists who are interested in a computer's working.
1. STRUCTURE OF A MODEL COMPUTING MACHINE

Compilation of programs normally means translation of high level language programs into the proper language of a computing machine. Therefore it is useful at first to study the structure and language of an average computer.

The main physical parts of a computer are the working storage (memory), the instruction processing unit and the data processing (arithmetic) unit.

The working storage consists of numbered (addressed) cells, numbered from 0 to a maximal address M. The content of the cell with the address $\delta$ is denoted by $\text{cont}(\delta)$. A cell consists of $N$ bits which can have the values 0 or 1.

$$
\begin{array}{cccc}
0 & 0 & \ldots & 0 \\
0 & 1 & \ldots & 1 \\
1 & 1 & \ldots & 1 \\
\end{array}
$$

The content of a cell, a binary word, is interpreted in mainly two different manners:

a) as an instruction by the instruction processing unit,
b) as a datum by the arithmetic unit.

The left $N-K$ bits form the so called operational part of

$$
\begin{array}{cccc}
\ldots & \ldots \\
N & K+1 & K & 1 \\
\end{array}
$$

operation address

or

part \omega operand

or

part $\delta$

the instruction to be executed. The right $K$ bits contain an address or an operand which the instruction refers to or operates with.

b) Possible interpretations as a datum are fixed point numbers, floating point numbers, logical values, and add-
resses. The exact representations as binary words are unim-
portant. Normally, the leftmost bit gives the sign of a num-
ber (0 for +, 1 for −), addresses are represented as non-ne-
gative fixed point numbers, true and false as fixed point 1
resp. 0.

The instruction processing unit has two main registers,
the instruction register (IR) and the instruction counter
(IC); the most important register of the arithmetic unit is
the accumulator (AC). IR, IC, and AC have lengths N,K,N re-
spectively.

The computer roughly works as follows:

1: The instruction processing unit transfers into the IR
the content of that cell denoted by the address α in the
IC.

2: The content of the IR is interpreted as an instruction
ω|β.

3a: In case of a so called jump instruction ω only a new
address α' for the IC is determined, and the machine con-
tinues with 1.

3b: In case of a so called arithmetic instruction ω it is
given to the arithmetic unit, and the machine continues
with 4.

4: Depending on the instruction ω the arithmetic unit works
as follows:

a: the datum cont(β) or the operand β is transferred in-
to the AC, or

b: the datum cont(β) operates on the content of the AC
and yields a new content for the AC, or

c: the content of the AC is transferred into the memory
so that it becomes cont(β).

5: The address α in the IC is increased by one, and the
machine continues with 1.
We give a list of the most important instructions; \( \beta \) is a variable which holds the position for a number (address or operand):

a) **arithmetic instructions:**

<table>
<thead>
<tr>
<th>transfer instructions</th>
<th>assembler language notation</th>
<th>meaning</th>
<th>high assembler language notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA ( \beta )</td>
<td>load the AC with ( \text{cont}(\beta) )</td>
<td>( \text{AC} := A\beta )</td>
<td></td>
</tr>
<tr>
<td>STA ( \beta )</td>
<td>store the content of the AC so that it becomes ( \text{cont}(\beta) )</td>
<td>( A\beta := \text{AC} )</td>
<td></td>
</tr>
<tr>
<td>ENA ( \beta )</td>
<td>enter ( \beta ) in the AC</td>
<td>( \text{AC} := \beta )</td>
<td></td>
</tr>
</tbody>
</table>

In order to differ between the instructions \( \text{AC} := A\beta \) and \( \text{AC} := \beta \), we put an \( A \) in front of \( \beta \) expressing that \( \beta \) means an address resp. fixed point number.

**fixed point operations**
ADA B  add cont(B) and the content of the AC and keep the result in the AC
    \[ AC := AC + f_i \beta \]
SBA B  subtract
    \[ AC := AC - f_i \beta \]
MUA B  multiply
    \[ AC := AC \times f_i \beta \]
DVA B  divide
    \[ AC := AC / f_i \beta \]

Floating point operations
FAD B  add
    \[ AC := AC + f_i \beta \]
FSB B  subtract
    \[ AC := AC - f_i \beta \]
FMU B  multiply
    \[ AC := AC \times f_i \beta \]
FDU B  divide
    \[ AC := AC / f_i \beta \]

Logical operations
AND B  logically conjunct cont(B) and the content of the AC
    \[ AC := AC \wedge \beta \]
OR B   disjunct
    \[ AC := AC \vee \beta \]
AUT B  exclusively disjunct
    \[ AC := AC \oplus \beta \]

b) Jump instructions:

UJP B  \( B \) becomes the new content of the IC
    \[ \text{jump to L} \beta \]

L is put in front of \( B \) in order to indicate that \( B \) is used as an address of an instruction (label) not of a datum.

AZJ,GR B  enter \( B \) into the IC if the content of the AC is greater than zero, otherwise increase the content of the IC by one
    \[ \text{if } AC > 0 \text{ then jump to } L \beta \]
AZJ,GE B
    \[ \text{if } AC > 0 \text{ then jump to } L \beta \]
AZJ,LS B
    \[ \text{if } AC < 0 \text{ then jump to } L \beta \]
AZJ,LE B
    \[ \text{if } AC < 0 \text{ then jump to } L \beta \]
AZJ,EQ B
    \[ \text{if } AC = 0 \text{ then jump to } L \beta \]
AZJ,NE B
    \[ \text{if } AC \neq 0 \text{ then jump to } L \beta \]
We assume that for floating and fixed point 0 exactly one representation is allowed, namely 00....0. Then, the conditional jumps work independently of the interpretation of the content of the AC as a floating or fixed point number.

\[
\text{STP operating of the machine ends properly or defined as we may say}
\]

If an address \( \beta \) is not in the range of 0 to \( M \) or if \( \omega \) cannot be interpreted as an instruction, operating of the machine ends erroneous or undefined as we may say.

2. THE NOTIONS PROGRAM AND PROGRAMMING LANGUAGE

In order to define programs for our model machine and to compare programs we divide the memory in an instruction and a data memory and we do not allow the arithmetic unit to read or to write in the instruction memory and the instruction processing unit to read in the data memory.

A \textit{machine code program} is a sequence of \( D \) \( N \)-bit words in the instruction memory. When a starting address \( a_0 \) in the instruction counter and data words in the data memory are given, operating of the machine is uniquely determined. Either it stops properly through a STP instruction or it ends undefined or it never finishes to work.

As it is rather awkward to read binary words and to figure out their meanings it is better to use (high)
assembler code notation for the instructions. A (high)
assembler code program is a sequence of labelled or unla-
belled instructions separated by ;. A labelled instruction
has the form $\beta:i$ (resp. $L\beta:i$) where $\beta$ is a number within the
range 0 to $D-1$.

The $\beta+1$st instruction in the sequence may be labelled by
$\beta$: (resp. $L\beta:$) and is assumed to be the content of cell $\beta$ in
the instruction memory. If $l$ is the length of the sequence
then all cells of the instruction memory, the addresses of
which are greater than $l-1$, are assumed to contain the stan-
dard undefined instruction $\emptyset$. The set of all (high) assemb-
ler code programs is a set of strings of symbols and forms
a so called (high) assembler language.

The meaning or interpretation $\phi(T)$ of a program $T$ is
obviously a transformation of storage states or storage con-
figurations. A state or configuration is given by a content
of the data memory and a content of the instruction counter.
We may write

$$\phi(T)|S \rightarrow S$$

where $S$ is the set of all states. $\phi(T)$ is, in general, only
partially defined because the program may not finish proper-
ly for certain starting states.

Every programmer knows that the same task can often be
done by different programs; he says the programs are equiv-
alent. Formally:

$$T_1 \cong T_2 \equiv Df \phi(T_1) = \phi(T_2)$$

But this definition of equivalence is very strong, we speak
of strong equivalence. Only few different programs are
strongly equivalent as the reader will easily find out. More
important is a weaker notion of equivalence with respect to
a generally accepted input function $I$ and an output function
$O$. $I$ yields for a given sequence $i$ of input data a starting
state $I(i) \in S$. $O$ yields for a given state $s \in S$ a sequence $O(s)$
of output data. Two programs $T_1$ and $T_2$ are called equivalent
if for all sets of input data i

\[(\star) \quad O(\phi(T_1)(I(i))) = O(\phi(T_2)(I(i)))\]

holds. This equation shall include the condition: If one side is undefined so the other is undefined, too, and vice versa. Equivalent programs $T_1$ and $T_2$ induce the same mapping of input data in output data.

The symbol $\phi$ in an expression like $(\star)$ is often dropped if context makes clear whether a program as a string $T$ of symbols or its interpretation $\phi(T)$ is meant. $(\star)$ could be written as follows:

\[O(T_1(I(i))) = O(T_2(I(i))).\]

One concrete definition of $I$ and $O$ can be given e.g. in the following way: We divide the data memory from $D$ to $M$ into an input data buffer from $D = D_I$ to $D_0 - 1$, an output data buffer from $D_0$ to $D_R - 1$, and an intermediate results memory from $D_R$ to $M$. Let a sequence $i = i_1, i_2, \ldots, i_n$ with $n < D_0 - D_I$.
of input data be given. Then I writes \( i_1 \) in cell \( D_1 \), \( i_2 \) in cell \( D_1 + 1 \), etc. The rest of the whole memory is filled with a standard value \( \Lambda \). The IC gets the initial address zero. \( O \) produces the output sequence

\[
O_1 = \text{cont}(D_0), \quad O_2 = \text{cont}(D_0 + 1), \ldots, O_{D_R - D_0} = \text{cont}(D_R - 1).
\]

The final address in the IC is simply dropped, nobody is interested in it.

We can see: A programming language \( L \) does not only consist of a set \( P \) of strings \( T \), called programs, but also of a mapping \( \phi \) which defines the meaning \( \phi(T) \) of every program \( T \), of an input function \( I \), and of an output function \( O : L = (P, \phi, I, O) \). The rules defining the set \( P \) of programs are called the syntax of the programming language \( L \), the rules defining \( \phi, I, O \) are called the semantic of \( L \). Above we have given syntax and semantic of a (high) assembler language for our model machine.

Every programmer observes that he often is not interested in the explicit instruction or data addresses. It is a great advantage for him to use freely choosable notations instead of numbers \( B \) with a fixed meaning. Normally, so-called identifiers are used, strings of letters and digits beginning with at least one letter. Our high assembler language may be extended in such a way which allows to use identifiers instead of instruction addresses \( _L\beta \) and data addresses \( _A\beta \). An additional requirement is that every identifier in a jump instruction must also occur as a label in front of an instruction.

The meaning of a program \( T_e \) of our extended high assembler language is given by the meaning of a program \( T_u \) of the unextended language which results from \( T_e \) if identifiers are replaced by instruction addresses \( _L\beta \) with \( 0 \leq \beta < D_1 \) and data addresses \( _A\beta \) with \( 0 \leq \beta < D_R \). Different identifiers must be replaced by different addresses, identical identifiers by iden-
tical addresses. From one and the same program $T_u$ may result different programs $T, T'$. But $T$ and $T'$ are equivalent and induce the same input-output function since only addresses of the intermediate result memory are permuted.

3. HIGH LEVEL PROGRAMMING LANGUAGES

We restrict our discussion on high level programming languages to so called ALGOL-like languages as ALGOL 60, FORTRAN, EULER, or ALGOL 68. A language like LISP 1.5 is not an ALGOL-like language, mainly because here we do not have the concept of an assignment statement.

Rather powerful high level programming languages can already be generated by a simple scheme like this:

1. We are given a set of certain strings of symbols called elementary statements $T_1, T_2, T_3, \ldots \ldots$.

2. We are given a set of certain strings called conditions $B_1, B_2, B_3, \ldots \ldots$.

3. There are five additional symbols $;;$if, then$,$else$,$fi$.

We have three rules to form statements inductively:

I. Every elementary statement is a statement.

II. If $\tau_1$ and $\tau_2$ are statements, then the string $\tau_1 ; \tau_2$ is a statement too (composed statement).

III. If $\tau_1$ and $\tau_2$ are statements and $B_i$ is a condition, then if $B_i$ then $\tau_1$ else $\tau_2$ fi is a statement (conditional statement).

There may be some additional rules which select from the set of statements those which are called programs of the special language which we have in view.

Under the assumption that the interpretations $\phi(T_i)$, $\phi(B_i)$ of the elementary statements and conditions is known, the interpretation of the programs must be defined. Our general idea is that a program $\tau$ is a notation for a trans-
formation \( \phi(\tau) \) of a set \( S \) of **abstract states**, in concreto
storage states, e.g.

\[
\phi(\tau) | S \rightarrow S
\]

A condition \( B_i \) is a notation for a condition

\[
\phi(B_i) | S \rightarrow \{ \text{true}, \text{false} \}
\]
on storage states. Let a non-elementary statement \( \tau \) be

\[
\tau = \tau_1 ; \tau_2
\]
or \( \tau = \text{if } B_i \text{ then } \tau_1 \text{ else } \tau_2 \text{ fi} \)
given. Then we define recursively for any state \( s \) in \( S \)

\[
(\#) \quad \phi(\tau)(s) = \text{Def} \quad \phi(\tau_2)(\phi(\tau_1)(s)) = (\phi(\tau_2) \circ \phi(\tau_1))(s)
\]

resp. \( \phi(\tau)(s) = \text{Def} \quad \begin{cases} 
\phi(\tau_1)(s) & \text{if } \phi(B_i)(s) = \text{true} \\
\phi(\tau_2)(s) & \text{if } \phi(B_i)(s) = \text{false} \\
\text{undefined} & \text{otherwise}
\end{cases} \)

\( \phi(\tau) \) is only partially defined; in case of a composed statement \( \phi(\tau) \) is undefined if \( \phi(\tau_1)(s) \) is undefined, or \( \phi(\tau_1)(s) \) is defined and \( \phi(\tau_2)(\phi(\tau_1)(s)) \) is undefined; in the case of a conditional statement \( \phi(\tau) \) is undefined if \( \phi(B_i)(s) \) is undefined, or if \( \phi(B_i)(s) \) is defined and \text{true} and \( \phi(\tau_1)(s) \) is undefined, etc.

We should give the following remark concerning the notation of function values: For state transformations as for many other functions we have used **prefix standard notation** for transformation values. We have written

\[
(s) \phi(\tau)
\]
similar to \( \text{sin}(5) \). Now, in (\#) we see that the \text{;} acts like composition of transformations; unfortunately, the sequence of transformations is reversed. Therefore, **postfix standard notation** for state transformation values

\[
(s) \phi(\tau)
\]
is more advantageous and the definitions now look as follows:

\[
(s)\phi(\tau) =_{Df} ((s)\phi(\tau_1)) \phi(\tau_2) = (s)(\phi(\tau_1) \circ \phi(\tau_2))
\]

resp. \( (s)\phi(\tau) =_{Df} \begin{cases} 
(s)\phi(\tau_1) & \text{if } \phi(B_i)(s) = \text{true} \\
(s)\phi(\tau_2) & \text{if } \phi(B_i)(s) = \text{false} \\
\text{undefined} & \text{otherwise}
\end{cases} \)

We already said that we may drop the symbol \( \phi \) if context makes clear whether a string or its interpretation is meant. Therefore, we simply have

\[
(s)\tau =_{Df} ((s)\tau_1) \tau_2 = (s)(\tau_1 \circ \tau_2)
\]

\[
(s)\tau =_{Df} \begin{cases} 
(s)\tau_1 & \text{if } B_i(s) = \text{true} \\
(s)\tau_2 & \text{if } B_i(s) = \text{false} \\
\text{undefined} & \text{otherwise}
\end{cases}
\]

It is not yet clear that \( \phi \) is a well defined function. A statement \( \tau_1;\tau_2;\tau_3 \) can be formed by two essentially different sequences of applications of the rule II:

\[
(\tau_1;\tau_2);\tau_3 \quad \text{and} \quad \tau_1;(\tau_2;\tau_3)
\]

As composition of transformations behaves associatively

\[
(s)((\phi(\tau_1) \circ \phi(\tau_2)) \circ \phi(\tau_3)) =
(s)(\phi(\tau_1) \circ (\phi(\tau_2) \circ \phi(\tau_3)))
\]

\( \phi \) is single valued, i.e. a well defined function. Associativity is not so clear if composing of statements does not simply mean composition of state transformations as we shall see later.

4. **ELEMENTARY STATEMENTS**

We specialize the states which ALGOL-like programs are assumed to work on. It is assumed that there is a large abstract data memory of cells with a (universal)
set $A_u$ of addresses. The cells may contain values in a set $V$. Furtheron, it is assumed that at any moment only a partial set $A_u \sqsubseteq A_u$ is assigned to a program. So the storage states are mappings

$$s \mid A \rightarrow V,$$

partial mappings, in general, as some cells in $A$ might not have been assigned any value. The set $S$ of all states is

$$S = \text{Df} \{ s \mid A \rightarrow V \}.$$ 

If we program in machine code the whole data memory is available, i.e. $A$ is always equal $A_u$ and every cell always contains a value. When programming in higher programming languages it is convenient to have a more general concept of storage states.

The interpretations $\phi(T_i)$ of the elementary statements are state transformations of the following types:

a) storage allocation $t^A_a$  
b) assignment of a value $t^V_a,f$  
c) storage freeing $t^F_a$

t$^A_a$ and $t^V_a$ have one parameter, an address $a$ in $A_u$. $t^V_a,f$ has two parameters, an address $a$ in $A_u$ and a partially defined so called value function $f \mid S \rightarrow V$.

These transformations are defined in the following way: Let $s \mid A \rightarrow V$ with $A \sqsubseteq A_u$ be a storage state.

a) $(s)t^A_a$ is defined to be the following storage state $s' \mid A' \rightarrow V$:

$$A' = \text{Df} \ A \cup \{a\}$$

$$s'(x) = \begin{cases} s(x) & \text{if } x \in A \\ \text{undefined} & \text{if } x = a \end{cases}$$

$(s)t^A_a$ is undefined if $a \in A$. Roughly: the argument domain $A$ of $s$ is enlarged by one address $a$. 

b) \((s \cdot t^V_{a,f})\) is defined to be the following storage state\
\[s' | A' + V:\]
\[
A' = \text{def} \ A \\
s'(x) = \begin{cases} s(x) & \text{if } x \neq a \\ f(s) & \text{if } x = a \end{cases}
\]

\((s \cdot t^V_{a,f})\) is undefined if \(a \notin A\). Roughly: \(s\) is changed only a little, only the address \(a\) is assigned a new value \(f(s)\), all other addresses keep their old values.

c) \((s \cdot t^F_a)\) is defined to be the following storage state\
\[s' | A' + V:\]
\[
A' = \text{def} \ A \setminus \{a\} \\
s'(x) = \text{def} \ s(x) \text{ for } x \in A'
\]

\((s \cdot t^F_a)\) is undefined if \(a \notin A\). Roughly: The argument domain \(A\) of \(s\) is made smaller by taking away one address \(a\).

For completeness it is convenient to have the following additional state transformations:

d) the identical transformation \(t^I\) with \((s \cdot t^I) = \text{def} \ s\)

e) the empty transformation \(t^\emptyset\) which is undefined for all states \(s\) in \(S\).

We have to clarify how these transformations are denoted in high level programming languages.

First, there are available denotations \(\phi\) for addresses. We also call them variables. Let us assume for a while that these denotations are standard denotations, i.e. denotations with a fixed meaning \(\phi(\beta)\). Storage allocation is then denoted by so called declarations

\[
\begin{align*}
\text{new } \beta & \quad \text{as in EULER} \\
\text{or real } \beta & \quad \text{as in ALGOL 60.}
\end{align*}
\]

We have

\[
\begin{align*}
\phi(\text{new } \beta) &= \text{def} \ t^A_{\phi(\beta)} \\
\phi(\text{real } \beta) &= \text{def} \ t^A_{\phi(\beta)}
\end{align*}
\]
real contains the additional information that cell β is a cell which can accept real numbers and real numbers only.

Secondly, there are available standard denotations for certain values, e.g.

1.56 for the corresponding real number,
46 for the corresponding integer number,
true for the corresponding logical value.

Furthermore, there are available some operator symbols such as + - * / \ \vee \neg \sin \abs

with a fixed and known meaning. With denotations for values as argument symbols, operators as function symbols, round brackets () and the comma sign formulas or terms can be formed which can be used to denote values, too, e.g.

ϕ(5+6*7) is the integer number 47
ϕ(true \wedge (6<5)) is the logical value false
ϕ(sin(5)+1.5) is a real number

The set of formulas can be enlarged if we also admit denotations for variables as argument symbols. Then, we are able to denote non-trivial value functions in the following way. Let e.g.

β + 15

be a formula. It denotes the following value function

ϕ|S→V

\[ \begin{array}{c}
s+ϕ(s) =_{Df} ϕ(s(β)+15) \\
\end{array} \]

Formulas without variable denotations not only denote values, but also trivial, i.e. constant value functions. In general, we write ψ[β₁,...,βₙ] for a formula ψ with the variable denotations β₁,...,βₙ occurring in ψ. ψ(β) is the value function

ϕ|S→V

\[ \begin{array}{c}
s+ϕ(s) =_{Df} ϕ(ϕ[s(β₁),...,s(βₙ)]) \\
\end{array} \]

where the βᵢ have been replaced by the value denotations s(βᵢ).
Now, assignments are denoted by so called assignment statements

\[ \beta := \psi \]

where \( \beta \) is a variable denotation and \( \psi \) is a value function denotation. \( \phi(\beta:=\psi) \) is

\[ \forall \phi(\beta), \phi(\psi) \] .

In high programming languages we normally do not have explicit notations for storage freeings. These are implicitly expressed by use of an end symbol as we shall see later.

The identical transformation is denoted by the so called dummy statement, an empty string. The empty transformation can normally not be expressed directly. We agree to denote \( t^\emptyset \) by \( \emptyset \).

5. REPLACEMENT OF ASSIGNMENT STATEMENTS BY THREE ADDRESS INSTRUCTIONS

Assignment statements may look quite different from high assembler code instructions as assignment statements may have very general formulas on their right hands. Therefore, general assignment statements cannot be interpreted by a computer immediately. We have to translate high level language programs \( \tau_1 \) into high assembler language programs \( \tau_2 \). We do so by replacing assignment statements by other statements. If replaced statements are strongly equivalent then so are \( \tau_1 \) and \( \tau_2 \) which can easily be seen. It is obvious that correct translation means that the meaning of programs is preserved.

In order to simplify our considerations we assume that all our formulas are made of binary infix operators as function symbols and variable denotations as argument symbols. Bracketing shall be complete and not superfluous; we shall call these formulas to be normal.

In a precise manner, normal formulas can be constructed by the following rules:
1. We are given a finite set \( O = \{ o_1, o_2, \ldots \} \) of binary infix operator symbols.

2. We are given a finite or enumerable set \( X = \{ x_1, x_2, \ldots, y_1, y_2, \ldots \} \) of argument symbols, in our special case variable denotations.

3. We are given two bracket symbols ( and ).

There are two rules to form normal formulas:

I. Any argument symbol is a normal formula.

II. If \( \psi_1 \) and \( \psi_2 \) are normal formulas and \( o_i \) is an operator symbol, then, the string \( (\psi_1 o_i \psi_2) \) is a normal formula, too.

If the interpretation \( \phi(o_i) \) of operator symbols as binary functions is known and if the interpretation \( \phi(x_i) \) of argument symbols as values is known, then, the interpretation (semantic) of normal formulas is clear, e.g.

\[
\phi(((5-4)+3))
\]

is the number 4.

A further simplification of our considerations is that we do not immediately translate into single address instructions like

\[
\begin{align*}
A\beta &: = AC, \\
AC &: = A\beta, \\
AC &: = AC \circ A\beta,
\end{align*}
\]

where \( o \) stands for an operator and \( \beta \) for a number (characterizing an address). Instead, we translate into three address instructions like

\[
\begin{align*}
\alpha &: = \gamma o \delta, \\
\alpha &: = \gamma,
\end{align*}
\]

where \( \alpha, \gamma, \) and \( \delta \) stand for variable denotations. For three address instructions no special register like an accumulator is necessary.

Before we present a complete translation process we show its idea of working by an example assignment statement
\[ \tau_1 = x := ((a \times b) + (c \times d)). \]

The process works in three steps plus one final step as our formula

\[ F = ((a \times b) + (c \times d)) \]

on the right hand has three operator symbols. In the first step we elect the innermost bracket pair \((a \times b)\), we allocate an auxiliary variable \(h_1\), and we generate a three address instruction

\[ \tau^A_{h_1}; \ h_1 := a \times b; \]

The remaining assignment statement

\[ x := (h_1 + (c \times d)) \]

is handled in the same manner. So, in the second step we generate

\[ \tau^A_{h_2}; \ h_2 := c \times d; \]

with the remaining assignment statement

\[ x := (h_1 + h_2). \]

The third step needs no new auxiliary variable, because auxiliary variables are "free" as soon as they occur on the right hand of three address instructions. We only generate

\[ h_1 := h_1 + h_2; \]

with the remaining assignment statement

\[ x := h_1. \]

A final step generates

\[ x := h_1; \ \tau^F_{h_2}; \ \tau^F_{h_1}. \]

The complete translation of \(\tau_1\) delivers the following composed statement \(\tau_2:\)

\[ \tau^A_{h_1}; \ h_1 := a \times b; \]

\[ \tau^A_{h_2}; \ h_2 := c \times d; \]

\[ h_1 := h_1 + h_2; \]

\[ x := h_1; \ \tau^F_{h_2}; \ \tau^F_{h_1}. \]
The storage freeings $t^{h_2}_{h_1}$ and $t^{h_2}_{h_1}$ at the end are necessary to assure equivalence of $\tau_1$ and $\tau_2$. Nevertheless, $\tau_1$ and $\tau_2$ are not strongly equivalent, but they are equivalent in the following sense: Let $s$ be a storage state

$$s|A\rightarrow V,$$

where $x,a,b,c,d$ denote addresses in $A$; $h_1,h_2$ denote different addresses in $A\setminus A$. Then the equation

$$(s)\phi(\tau_1)=(s)\phi(\tau_2)$$

holds, by which weak equivalence of translated programs is assured.

Now we define the complete translation process: Let an assignment statement

$$x:=F$$

be given, where $F$ is a normal formula. The process works in $r+1$ steps if $r_o$ is the number of operator symbols occurring in $F$. At the beginning of the $n$-th step the process operates with a series of $m_{n-1}$ allocated auxiliary variables,

a partial series of $k_{n-1}$ free auxiliary variables,

a piece of program $\Pi_{n-1}$ of storage allocations

and $n-1$ three address instructions,

a remaining assignment statement $x:=F_{n-1}$.

At the beginning no auxiliary variable is allocated, $\Pi_0$ is empty and $F_0$ is $F$.

The transition from the $n$-1st step to the $n$-th step ($1\leq n \leq r+1$) has the following partial actions:

1) In $F_{n-1}$ we look for an innermost bracket pair

$$F_{n-1}=G_{n-1}(vw)F_{n-1}$$

where $v$ and $w$ are variable denotations. If there is none, $F_{n-1}$ is a variable only and we are ready. We assign $x:=F_{n-1}$ and all auxiliary variables are freed.

2) If $v$ or $w$ are auxiliary variables they are declared to be free.

3) If there is no free auxiliary variable, then, there must, otherwise, there may be allocated a very new free auxiliary variable.

4) a free auxiliary variable hv is taken and the instruc-
5) hv replaces (vow) in $F_{n-1}$. We get

$$F_n = D_f G_{n-1} hv H_{n-1},$$

hv is declared not to be free.

6) Go on with 1).

We see: Immediately before the n-th step the auxiliary variables occurring in $F_{n-1}$ are exactly those which the process has declared to be non-free. No auxiliary variable in $F_{n-1}$ occurs twice or more times.

It is obvious that a good translation process is one which needs few auxiliary variables. So, we make the following refinement:

3') No new auxiliary variable is allocated if there is a free one.

and we can state: Let a formula $F$ and a fixed successive election of innermost bracket pairs be given. Then, with 3') the number of used auxiliary variables is not greater than without 3').

It is clear that during the translation process a more or less complicated bookkeeping for the free and non-free auxiliary variables is necessary. In order to simplify this bookkeeping we assume that there is an array

$$h[1], h[2], \ldots \ldots$$

of auxiliary variables available and we always allocate that new auxiliary variable with the lowest index. So, the indices for the allocated variables for $F_n$ form an interval

$$[1:m_n] \subseteq \mathbb{N}$$

of natural numbers. For the non-free auxiliary variables we want to have an interval

$$[1:k_n] \subseteq [1:m_n].$$

So, for bookkeeping only two pointers to $k_n$ and to $m_n$ are necessary.
In order to force our translation process to go in this direction, we have to make at least the following necessary refinement:

4') take for hv that free auxiliary variable with the lowest index.

Unfortunately, this refinement is not sufficient as the following example shows:

\[ F = F_0 = (((a\times b) + ((a\times d) \times c)) / ((a\times a) + (b\times b))) \]
\[ F_1 = (((a\times b) + (h[1] \times c)) / ((a\times a) + (b\times b))) \]
\[ F_2 = ((h[2] + (h[1] \times c)) / ((a\times a) + (b\times b))) \]
\[ F_3 = ((h[2] + (h[1] \times c)) / (h[3] + (b\times b))) \]
\[ F_4 = ((h[2] + h[1]) / (h[3] + (b\times b))) \]
\[ F_5 = (h[2] / (h[3] + (b\times b))) \]

After the 5th step three auxiliary variables h[1], h[2], h[3] are allocated, h[1], h[3] are not free, the indices {1,3} do not form an interval within \([1:3] \subset \mathbb{N}\).

As the actions 2), 3), 4), 5) are deterministic, we see the only chance to achieve our goal in making action 1) deterministic.

1') Elect the leftmost innermost bracket pair (vow) in \(F_{n-1}\).

If 1'), 3'), 4') are respected, we may prove the following

**Theorem on the pushdown principle** for intermediate results: The indices of the non-free auxiliary variables in \(F_n\) form an interval \([1:k_n] \subset [1:m_n]\). The non-free auxiliary variables \(h[1], \ldots, h[k_n]\) occur in the sequence of their indices in \(F_n\) and they occur left of the leftmost closing bracket in \(F_n\).

This Theorem has the following consequence for the translated programs at execution time: Those intermediate results which have been computed and stored last are reused first (last in - first out). K. Samelson and F.L. Bauer observed
this pushdown principle in 1959. The translation process applied to

\[ x := \frac{((a \times b) + ((a \times d) \times c))}{((a \times a) + (b \times b))} \]

gives the following auxiliary variable numbers and program

<table>
<thead>
<tr>
<th>n</th>
<th>k_n</th>
<th>m_n</th>
<th>h[1]</th>
<th>h[2]</th>
<th>h[3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>a \times b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>a \times d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>h[2] = h[2] \times c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>h[2] = a \times a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>3</td>
<td>h[3] = b \times b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td></td>
<td>x = h[1]</td>
<td></td>
</tr>
</tbody>
</table>

For simplicity we have dropped all storage allocations and freeings.

6. TRANSLATION METHODS DIFFERENT FROM THE PUSHDOWN METHOD AND FORMULAS ON THE NUMBER OF AUXILIARY VARIABLES

The first endeavours to evaluate formulas mechanically by real computers were made in 1952 by H. Rutishauser. He associated so called level numbers with every opening and closing bracket in a formula \( F \)

\[ (((a \times b) + ((a \times d) \times c)) / ((a \times a) + (b \times b))) \]

and in action 1) of our general translation process he elected innermost bracket pairs with top level numbers. This method is well justified as we have learnt, but it has several disadvantages compared with the pushdown method:

1) The intermediate results do no longer occur in the fashion "last in - first out".
2) The whole formula must be read before an innermost bracket pair on top level can be determined.

3) The formulas must be read again and again from the very beginning, whereas the pushdown method needs to read only once from left to right.

For his first FORTRAN compiler J.W. Backus used a method which is in some sense similar to that of Rutishauser.

Neither the pushdown method nor Rutishauser's method is optimal with respect to the number of necessary auxiliary variables. A look at the general translation process shows that any translation of a given formula \( F \) needs at most \( r(F) \) auxiliary variables where \( r(F) \) is the number of operators occurring in \( F \). If the process respects 3'), then, at most \( s(F) \) auxiliary variables are necessary where \( s(F) \) is the number of innermost bracket pairs in \( F \). The relation

\[
s(F) \leq 1/2(r(F)+1)
\]

can easily be shown.

The number \( k(F) \) of auxiliary variables which the pushdown method needs can be calculated recursively:

\[
k(F) = \begin{cases} 
0 & \text{if } F \text{ is a variable} \\
k(F_1) & \text{if } k(F_1) > k(F_2) \\
k(F_2)+2 & \text{if } 0 < k(F_1) \leq k(F_2) \text{ or } 0 = k(F_1) = k(F_2) \\
k(F_2) & \text{if } k(F_1) = 0, \ k(F_2) > 0 
\end{cases}
\]

\( k((F_1 \circ_i F_2)) = \)

If we define the nesting

\[
n(F) = \begin{cases} 
0 & \text{if } F \text{ is a variable} \\
\max(n(F_1), n(F_2)) + 1 & \text{if } F = (F_1 \circ_i F_2)
\end{cases}
\]

of a formula \( F \) then the relation

\[
k(F) \leq n(F)
\]

can easily be proven. The number \( o(F) \) of auxiliary variables which an optimal translation method needs can be calculated recursively, too:
\[ o(F) = \begin{cases} o(F_1) & \text{if } o(F_1) > o(F_2) \\ o(F_2) + 1 & \text{if } o(F_1) = o(F_2) \\ o(F_2) & \text{if } o(F_2) > o(F_1) \end{cases} \]

and the relation \( o(F) \leq k(F) \)

can be proven by induction. The pushdown method is not \( \varphi \)-optimal for every formula \( F \), e.g. for

\[ F = ((a-b)/((c-d)-(e/f))) \]

We have \( o(F) = 2 \) and \( k(F) = 3 \).

7. **Translation to Single Address Instructions**

As most machines are not three address but single address machines we now have to care for the translation of assignment statements \( x := F \) to single address instructions. We assume that beside the load and store instructions

\[ AC := \alpha \quad \alpha := AC \]

for every operator \( \circ \) two instructions

\[ AC := AC \circ \alpha \quad \alpha := \alpha \circ AC \]

are available. Here \( \alpha \) stands for a variable denotation.

The application of the pushdown method to the example assignment statement

\[ x := ((a \times b) + (c \times d)) \]

gives us a series of three address instructions:

\[
\begin{align*}
    h[1] & := a \times b; \\
    h[2] & := c \times d; \\
    x & := h[1].
\end{align*}
\]

The additional translation to single address instructions gives:

\[ AC := a; \quad AC := AC \times b; \quad h[1] := AC; \]
\[ AC := c; \quad AC := AC \times d; \quad h[2] := AC; \]
\[ AC := h[1]; \quad AC := AC + h[2]; \quad h[1] := AC \]
\[ AC := h[1]; \quad x := AC \]

It is clear that several load and store instructions could be saved:
\[ AC := a; \quad AC := AC \times b; \quad h[1] := AC; \]
\[ (\exists) \quad AC := c; \quad AC := AC \times d; \quad AC := h[1] + AC; \]
\[ x := AC \]

Therefore, the following question arises: Can these savings be done systematically by modifications of the pushdown translation process? The answer is yes. The modifications only touch the assembler code generation, the handling of auxiliary variables stays the same. The idea behind our proceeding is to keep the value of the last non-free auxiliary variable in the accumulator as long as possible.

The pushdown translation process is modified in the following way:

Case 1: If \( F \) is a variable, the instructions
\[ AC := F; \quad x := AC \]
are generated.

Case 2: Let \( F \) be not only a variable.

In step \( n=1 \) we generate
\[ AC := v; \quad AC := AC \times v; \]
(vow) in \( F=F_o \) is replaced by \( h[1] \) and we get \( F \). \( h[1] \) is considered not to be free.

In a step \( n>1 \) (vow) has one of the following forms:

a) \( (h[k] \times v) \)
b) \( (v \times h[k]) \)
c) \( (h[k-1] \times v \times h[k]) \)
d) \( (v \times v) \)

where \( v \) and \( w \) are non-auxiliary variables. We generate

a) \( AC := AC \times w; \)
b) \( AC := v \times AC; \)
c) \( AC := h[k-1] \times AC; \)
d) \( h[k] := AC; AC := v; AC := AC_ow; \)

and we replace \((vow)\) in \(F_{n-1}\) by

a) \( h|k| \)  

b) \( h|h| \)  

c) \( h|k-1| \)  

d) \( h|k+1| \)

In the final step we generate

\[ x := AC \]

These modifications deliver exactly the translation \((\ast)\) of our example assignment statement above.

If for an operator \( o \) only one instruction \( AC := AC_ow \), but not

\[ AC := aoAC \]

is available, which is normally true for subtraction and division, then, in case b) and c) different code must be generated:

b) \( h[k] := AC; AC := v; AC := AC_oh[k]; \)

c) \( h[k] := AC; AC := h[k-1]; AC := AC_oh[k]; \)

8. GENERAL INFIX FORMULAS AND THEIR TRANSLATION

Usually, people do not like to write normal formulas with all that bracketing. They wish to drop brackets and programming languages have to admit it. **General infix formulas** can be constructed by the following rules:

The symbols given are the same as for normal formulas. There are three rules to form general infix formulas or formulas for short:

I. Any argument symbol is a formula.

II. If \( \psi_1 \) and \( \psi_2 \) are formulas and \( o_i \) is a binary infix operator symbol, then, the string \( \psi_1 o_i \psi_2 \) is a formula, too.

III. If \( \psi \) is a formula, then, the string \((\psi)\) is a formula, too.

The interpretation of general formulas is ambiguous because \( f(\langle 5-4 \rangle + 3) \) can be the number 4 or \( f(5-\langle 4+3 \rangle) \) can be
the number \(-2\). In order to get an unambiguous interpretation of general formulas we associate with any formula \(\psi\) in an unambiguous manner a normal formula \(\psi_n\). The interpretation \(\phi(\psi)\) then is defined to be \(\phi(\psi_n)\).

The operator symbols are assigned priorities

\[ \text{Pri}\ 0 + Z \]

(and) get also a priority with

\[ \text{Pri}(\cdot) = \text{Pri}(\cdot) < \text{Pri}(o_i) \]

for all \(o_i \in O\). In order to define a relation "more normal" for formulas conveniently, we enclose formulas \(\psi\) in end symbols \(\text{\{\}\}}\) and define

\[ \text{Pri}(\text{\{\})} = \text{Pri}(\text{\{\}}) = \text{DF} \text{Pri}(\cdot). \]

**Definition:** Formula \(\psi'\) is called **more normal** than \(\psi\) (we write \(\psi \rightarrow \psi'\)) if and only if \(\psi'\) can be generated from \(\psi\) by one of the following two possibilities:

1. We choose any partial formula \((\psi)\) within \(\psi\) which is superfluously bracketed. Then we drop the outermost brackets of \((\psi)\), and so we get \(\psi'\).

2. Let in \(\text{\{\}\}}\) be a partial string

\[ o_0 o_1 o_1 o_2 o_2 \]

with:

a. \(o_1, o_2\) are argument symbols or bracketed formulas,

b. \(o_0 \in O \cup \{\cdot, \}\) and \(o_2 \in O \cup \{\cdot, \}\), but not \(o_0 = (\text{ and } o_2 = )\),

c. \(o_1 \in O\),

d. \(\text{Pri}(o_0) < \text{Pri}(o_1) > \text{Pri}(o_2)\).

Then we enclose \(o_1 o_1 o_2\) in round brackets, and so we get \(\text{\{\}\}}\).

It can be proven that successive applications of the relation \(\rightarrow\) always lead to a maximal formula \(\psi_n\). \(\psi_n\) is normal, uniquely determined by the starting formula \(\psi\), and independent of the sequence of \(\rightarrow\) applications. The number of steps is given by

\[ S(\psi) = \text{number } r \text{ of the occurrences of operator symbols in } \psi \]
number of the superfluously bracketed partial formulas in $\psi$

- number of the bracketed, but not superfluously bracketed formulas in $\psi$.

$S(\psi)$ is $\geq 0$, decreases by 1 in every step, and is 0 if and only if $\psi$ is normal. Example:

$\vdash \psi \downarrow = \vdash (5\times 6+(3/4\times 5)) + 3+2 \downarrow$
$\vdash 5\times 6+(3/4\times 5) + 3+2 \downarrow$
$\vdash 5\times 6+(3/4\times 5)) + 3+2 \downarrow$
$\vdash 5\times (6+(3/4\times 5)) + (3+2) \downarrow$
$\vdash (5\times (6+(3/4\times 5))) + (3+2) \downarrow$

$S(\psi) = 6 + 2 - 1 = 7$

Now, let an assignment statement

$x := \psi$

be given where $x$ is a variable denotation and $\psi$ a formula with variable denotations as argument symbols only. The correct translation process then is defined as follows: Firstly, the formula $\psi$ is to be normalized

$x := \psi_n \rightarrow x := \psi_n$. 

Secondly, the assignment statement $x := \psi_n$ is to be translated into three address instructions as is shown in Chapter 5

$x := \psi_n \rightarrow \Pi; x := v$. 

Here $\Pi$ is a series of three address instructions and $v$ is a variable.

We now wish to intermingle the normalization and the translation process and define the following relation $\vdash$:

Definition: $\Pi; x := \psi \vdash \Pi'; x := \psi'$ if and only if one of the following two conditions holds:

1. $\Pi = \Pi'$ and $\psi \vdash \Pi'$
2. In $\psi$ is chosen a partial formula (vow), where $v$
and w are variables or auxiliary variables. (vow) is replaced by a free auxiliary variable hv, so we get ψ'. A new three address instruction hv:=vow is generated which gives us Π'.

If we apply the relation ⊢ successively to a given assignment statement x:=ψ we end up with an equivalent program Π*; x:=v* independently of the way we have chosen. This program is maximal with respect to ⊢ and v* is a variable. The number of steps is given by

\[ S(ψ) = r + S(ψ) \]

S(ψ) is > 0, decreases by 1 in every step, and is 0 if and only if ψ is a variable.

As we are not really interested in the normal form \( ψ_n \) of ψ we try to define an equivalent translation process which avoids to introduce new brackets.

**Definition:** Π; x:=ψ ⊢ Π'; x:=ψ' if and only if one of the following conditions holds:

1. Choose any bracketed variable (v) within ψ and drop the brackets.
2. Let in ⊢ψ be a partial string
    \[ o_0 v o_1 v o_2 \]
    with
    a. v and w are variables,
    b. \( o_0 \in \{, \}, o_2 \in \{, \}, \)
    c. \( o_1 \in \emptyset, \)
    d. \( \text{Pri}(o_0) < \text{Pri}(o_1) \geq \text{Pri}(o_2). \)

Then vow is replaced by a free auxiliary variable which gives us ψ' and a new three address instruction hv:=vow is generated, giving us Π'.

Successive applications of ⊢ resp. ⊢ produce equivalent final results, for if v* is a variable, then, we can prove:

\[ Π; x:=ψ∗ ⊢ Π∗; x:=v∗ \text{ if and only if } Π; x:=ψ∗ ⊢ Π∗; x:=v∗. \]
If we always look for the leftmost occurrences of strings \( v \) or \( o_1 v o_{1} w_2 \), we get the same result as if we first normalize and then translate by the pushdown method for normal formulas. So, we have proven that there is even a pushdown method for general infix formulas which avoids to introduce new brackets.

Now we can write a translator program in an ALGOL-like language which translates assignment statements
\[
x := \psi;
\]
due to the pushdown method just defined. := and ; take the role of the end symbols \( \$$ \) and \( \)} \( \), i.e.
\[
\text{Pri}(::) = \text{Pri}(); = \text{df} \text{Pri}().
\]
We have an input store
\[
\text{string array } \chi[1: \infty],
\]
which initially is filled with the input assignment statement. The input pointer \( i \) points to the next symbol to be read. Furthermore, we have a pushdown store
\[
\text{string array } \kappa[1: \infty],
\]
which initially is empty. The pushdown pointer \( k \) points to the last relevant symbol in the pushdown. A program variable \( \Pi \) gathers the generated three address instructions.

\[
\begin{align*}
(1) & \quad \text{string array } \chi[1: \infty]; \\
(2) & \quad \text{string array } \kappa[1: \infty]; \\
(3) & \quad \text{integer } i; \\
(4) & \quad \text{integer } k; \\
(5) & \quad \text{string } \Pi; \\
(6) & \quad \text{string } \nu; \\
(7) & \quad \Pi := \nu; \\
(8) & \quad \kappa[1] := \chi[1]; \kappa[2] := \chi[2]; \\
(9) & \quad k := 2; i := 3; \\
(10) & \quad \text{pushdown: } \kappa[k+1] := \chi[i]; \\
(11) & \quad k := k + 1; \\
(12) & \quad \text{read: } i := i + 1; \\
(13) & \quad \text{repeat: if variable}(\chi[i]) \text{ then goto pushdown;}
(14) & \quad \text{if } \chi[i] = 9 \text{ then goto pushdown;}
\end{align*}
\]
if pri(χ[i]) > pri(χ[k-1]) then goto pushdown;

if χ[i] = 5 \& k[k-1] = 5 then
begin \kappa[k-1] := \kappa[k]; k := k-1; goto read end;

if χ[i] = 5 \& k[k-1] = 5 then
begin \Pi := \Pi \oplus k[k-2] \oplus \kappa[k-1] \oplus \kappa[k] \oplus 5; 9;
end;

hv := free auxiliary variable with lowest index;
\Pi := \Pi \oplus hv \oplus 5 \oplus \kappa[k-2] \oplus \kappa[k-1] \oplus \kappa[k] \oplus 5; 9;
\kappa[k-2] := hv;
k := k-2;
goto repeat;
end:

The effect of the translator program is demonstrated for the example assignment statement \( x := (b+c\times d)/(a-d) + b \); with

\[
\text{pri}(\cdot) = \text{pri}(;) = \text{pri}(\cdot) = \text{pri}(\cdot) = 0, \\
\text{pri}(+) = \text{pri}(-) = 1, \text{pri}(\times) = \text{pri}(\div) = 2.
\]

In a table the successive contents of the pushdown, the pushdown pointer, the input pointer, and of the program variable are shown

<table>
<thead>
<tr>
<th>( \kappa )</th>
<th>k</th>
<th>i</th>
<th>( \Pi )</th>
</tr>
</thead>
<tbody>
<tr>
<td>x:=</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>x:= (</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>x:= (b</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>x:= (b+</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>x:= (b+c</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>x:= (b+c</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>x:= (b+c\times d</td>
<td>8</td>
<td>9</td>
<td>h[1] := c\times d;</td>
</tr>
<tr>
<td>x:= (b+h[1]</td>
<td>6</td>
<td>9</td>
<td>h[1] := c\times d; h[1] := b+h[1];</td>
</tr>
<tr>
<td>x:= (h[1]</td>
<td>4</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>x:= h[1]</td>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>x:= h[1]/</td>
<td>4</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>x:= h[1]/(</td>
<td>5</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>x:= h[1]/(a</td>
<td>6</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>x:= h[1]/(a-</td>
<td>7</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>x:= h[1]</td>
<td>3</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>x:= h[1]+</td>
<td>4</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>x:= h[1]+b</td>
<td>5</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>x:= h[1]</td>
<td>3</td>
<td>18</td>
<td></td>
</tr>
</tbody>
</table>

9. LABEL DECLARATIONS AND JUMP STATEMENTS

In Chapters 3 and 4 we presented a concept how syntax and semantic of a programming language can be defined. Those programs were only linear programs because they did not include jumps. The interpretation \( \phi(\tau) \) of a linear program \( \tau \) was a transformation

\[
\phi(\tau)\mid S+S
\]

of a set \( S \) of storage states. In order to include jumps we must extend the set \( S \) by a set \( M \) of label states

\[
\phi(\tau)\mid S\times M+S\times M
\]

\((s,1)+(s,1)\phi(\tau)=(s',1')\)

We assume that \( M \) contains a specific label state \( \underline{\ell} \), the usual label state.

What types of interpretations \( \phi(T_i) \) of elementary statements do we want to have? By all means the old types of Chapter 4:

a) \( t_a^A \)  
b) \( t_a^V \)  
c) \( t_a^F \)  
d) \( t_I \)  
e) \( t_\emptyset \)

Additionally we want to have the

f) labelling \( t_m^L \) and

g) jumping \( t_m^J \)

where the parameter \( m \) is an unusual label state \( m \in M \setminus \{\underline{\ell}\} \).

These transformations are functions of two arguments. Therefore, they are principally different from those transformations which we became acquainted with earlier and which have only one argument. As we use postfix standard notation for transformations we are allowed to use the same symbol for different transformations in transformation value denotations. The number of arguments in

\((s)t_a^V \) or \((s,1)t_a^V \)

makes clear whether the old or the new transformation is meant.

We define the new transformations a) to e):

\[
(s,1)t =_{DF} \begin{cases} 
((s)t,1) & \text{if } 1 = \underline{\ell} \\
(s,1) & \text{if } 1 \not= \underline{\ell} 
\end{cases}
\]
For \( t \) we may put \( t^A, t^V, t^F, t^I, \) or \( t^0 \). We mention that 
\((s,1)t^0\) is not totally undefined contrary to \((s)t^0\). \((s,1)t^0\) 
is undefined if and only if \( 1 \) is the usual label state \( \sum \).
We define \( t^L_m \) and \( t^J_m \) with \( m \in M \setminus \{ \sum \} \):

\[
(s,1)t^L_m = \text{Df} \begin{cases} 
(s,\sum) & \text{if } 1 = m \\
(s,1) & \text{if } 1 \neq m 
\end{cases}
\]

\[
(s,1)t^J_m = \text{Df} \begin{cases} 
(s,m) & \text{if } 1 = \sum \\
(s,1) & \text{if } 1 \neq \sum 
\end{cases}
\]

In words: A state \((s,\sum)\) with the usual label states \( \sum \) means 
that we work as we usually do. \( \text{a)} - \text{e)} \) work as earlier, label-
ing \( f \) is neglected because \( m \neq \sum \), only jumping \( g \) changes 
the label state \( \sum \), namely to \( m \neq \sum \). From now on we are in an un-
usual label state and we are looking for a corresponding la-
belling \( t^L_m \). Only \( t^L_m \) changes the state \((s,m)\), namely to \((s,\sum)\).

How are the elementary statements denoted in commonly 
known programming languages? \( \text{a)} - \text{e)} \) are denoted as in earlier 
Chapter 4, e.g. assignment statements denote value assign-
ments with 

\[
\phi(\beta := \psi) = \text{Df} \ t^V_{\phi(\beta), \psi}.
\]

\( \beta \) is a denotation for a storage address \( \phi(\beta) \), a variable, \( \psi \) 
is a formula denoting a value function \( \phi(\psi) \mid S \rightarrow V \). Labellings 
\( f \) are denoted by so called label declarations \( u \) with 

\[
\phi(u:) = \text{Df} \ t^L_{\phi(u)}
\]

where at this stage we may assume \( u \) to be a standard deno-
tation for a label state \( \phi(u) \) in \( M \setminus \{ \sum \} \). Jumpings \( g \) are deno-
ted by so called jump statements \( \text{jumpto } u \) with 

\[
\phi(\text{jumpto } u) = \text{Df} \ t^J_{\phi(u)}.
\]

The semantic of non-elementary statements must be re-
defined also, because we do not want the semicolon \( ; \) act as 
a simple composition \( \odot \) of transformations. In general, we do 
not want the equation 

\[
(s,1)\phi(\tau_1 ; \tau_2) = (s,1)(\phi(\tau_1) \odot \phi(\tau_2))
\]

\[
= ((s,1)\phi(\tau_1))\phi(\tau_2)
\]
to hold. Fortheron, we must restrict the set of syntactically correct programs to those in which the same label declaration $\nu$: occurs at most once. This requirement is intuitively clear, otherwise, programs would work in a non-deterministic manner.

A non-elementary statement $\tau$ is of the form

$$\tau = \tau_1; \tau_2$$

or $\tau = \text{if } B_1 \text{ then } \tau_1 \text{ else } \tau_2 \text{ fi}.$

In case of a composed statement $\tau$ we define

$$(s,1)\phi(\tau) = \text{Df} \begin{cases} (s,1)(\phi(\tau_1)\circ\phi(\tau_2))\circ(\phi(\tau_1)\circ\phi(\tau_2))^n \\
\text{where } n \text{ is the least integer number } > 0 \\
\text{such that the label state } 1' \text{ in the result } (s',1') \text{ does not occur in any label declaration in } \tau_1 \text{ or } \tau_2 \\
\text{undefined if there is no such } n \end{cases}$$

The situation "undefined" may happen e.g. if $\tau$ contains backward jumps. $(s,1)\phi(\tau)$ is undefined then, because we run into an infinite loop. In case of a conditional statement we define:

$$(s,1)\phi(\tau) = \text{Df} \begin{cases} (s,1)\phi(\tau_1) \text{ if } 1 = \nu_1 \text{ and } \phi(B_1)(s) = \text{true} \\
\text{or } 1 \text{ occurs in a label declaration in } \tau_1 \\
(s,1)\phi(\tau_2) \text{ if } 1 = \nu_1 \text{ and } \phi(B_1)(s) = \text{false} \\
\text{or } 1 \text{ occurs in a label declaration in } \tau_2 \\
(s,1) \text{ if } 1 \notin \nu \text{ and } 1 \text{ does not occur in any } \\
\text{label declaration of } \tau_1 \text{ or } \tau_2 \\
\text{undefined otherwise} \end{cases}$$

This definition requires that the programmer may not write jump statements which lead out of any then statement into the corresponding else statement and vice versa. This requirement is not absolutely necessary, we could give a more elaborate definition which avoids it. ALGOL 60 does not know this restrictive requirement, but ALGOL 68 does.

Because double label declarations are forbidden $\phi$ is a well defined interpretation. Especially it can be proven that the $;$ acts associatively

$$(s,1)\phi(\tau_1;\tau_2;\tau_3) = (s,1)\phi(\tau_1;\tau_2;\tau_3)$$
The proof is not as trivial as in Chapter 3.

In order to show that these definitions work as we expect them to work, we calculate the results of

\[(s, \bar{\ell})\oplus (\text{jumpto } m; m:) \quad \text{and} \quad (s, \bar{\ell})\oplus (m; \text{jumpto } m)\]

\[(s, \bar{\ell})(\phi(\text{jumpto } m)\odot \phi(m:)) \text{ gives } (s, \bar{\ell}). \quad \bar{\ell} \text{ does not occur in any label declaration. Therefore} \]

\[(s, \bar{\ell})\oplus (\text{jumpto } m; m:) = (s, \bar{\ell})\]

as we expect. \((s, \bar{\ell})(\phi(m:))\odot \phi(\text{jumpto } m)) \text{ gives } (s, m) \text{ and } m \text{ occurs in } m:. \quad (s, m)(\phi(m:)\odot \phi(\text{jumpto } m)) \text{ gives } (s, m). \text{ Therefore} \]

\[(s, \bar{\ell})\oplus (m; \text{jumpto } m)\]

is undefined as we expect.

In assembler languages we have only rudimentary conditional statements, namely conditional jumps. Conditional statements

\[
\text{if } B_i \text{ then } \tau_1 \text{ else } \tau_2 \text{ fi}
\]

can be correctly translated in the following way:

\[
\text{AC} := B_i; \\
\text{if AC} = 0 \text{ then } \text{jumpto } m_1; \\
\tau_1; \\
\text{jumpto } m_2; \\
m_1: \\
\tau_2; \\
m_2:
\]

\(m_1\) and \(m_2\) are label denotations which do not yet occur in the program.

In order to get a nice storage allocation scheme for programs most of the well known programming languages proceed as follows.

1. The jump statements \text{jumpto } u \text{ are replaced by so called goto statements } \text{goto } u.
2. One new rule IV is added to those of Chapter 3:

IV. If \( \tau \) is a statement then the string \texttt{begin } \tau \texttt{ end} 
is a statement, too.

3. Further restrictions are imposed on programs:

a. The predecessor of a storage allocation must be a

\begin{verbatim}
begin or a ; preceded by another storage allocation.
\end{verbatim}

b. Storage freeings are not allowed to be written down.

c. Goto statements may not lead from outside in a

\begin{verbatim}
begin end statement which has a storage allocation
immediately behind \texttt{begin}. We call such statements
blocks.
\end{verbatim}

What interpretation is involved with \texttt{begin}, \texttt{end}, and
\texttt{goto}? \texttt{begin} acts as a dummy statement. \texttt{end}
implicitly frees all storage places which have been allocated shortly behind
the associated \texttt{begin}. A goto statement which leads out of
(eventually nested) blocks must implicitly free all storage
places which are freed by the associated \texttt{end}-s.

As a result storage allocation and freeing behave push-
down like: a storage place allocated last is freed first.

Example:

\begin{verbatim}
begin real a; real b; ...... 
n;; begin real c; real d;......; goto m;...end 
; ; 
m;; 
; ; 
goto n; 
; ; 
end
\end{verbatim}

If \texttt{goto m} would act as \texttt{jumpto m} it would not make sense to
enter the inner block again by \texttt{goto n}, because it is not
defined to allocate the storage place \texttt{c} a second time before
having freed it. See Chapter 4.
10. ARRAYS AND SUBSCRIPTED VARIABLES

If many storage places for values of the same mode are necessary, then, it is convenient to have a shorthand for such a storage allocation. In ALGOL 60 we write a so-called array declaration, e.g.

\[
\text{integer array } a[u:v],
\]

where \( u \) and \( v \) stand for integer number denotations with \( u \leq v \). \( a \) is a standard denotation for an array store

\[
a: \ldots \quad | \quad | \quad | \quad | \quad | \quad | \quad | \quad | \quad | \quad | \quad | \ldots
\]

\[
\underbrace{w \quad w+1 \quad \ldots \ldots \ldots \ldots \quad v}_{\text{allocated places}}
\]

A single address in this array can be characterized by a couple \((a,k)\) where \( k \) denotes a number with \( u \leq k \leq v \). In ALGOL 60 this address is denoted by \( a[k] \) and is called a subscripted variable. If, as above, array bounds are fixed numbers (here we speak of static arrays), then, an array declaration may be viewed as an abbreviation for finitely many storage allocations for simple variables, e.g.

\[
\text{real array } [1:10]
\]

is an abbreviation for

\[
\text{real } a[1]; \ldots ; \text{real } a[10]
\]

and

\[
\text{Boolean array } b[1:10,0:4]
\]

is an abbreviation for

\[
\text{Boolean } b[1,0]; \text{Boolean } b[1,1]; \ldots \quad \text{;Boolean } b[10,4].
\]

Since in practice in a computing machine only one linear data memory is available for all simple and for all subscripted variables we have to look for methods how to store arrays in this memory. Let a storage state \( s|A+V \) be given and let BFS denote the beginning of free storage.
For an array \texttt{integer array a[\mu: \nu], \mu \leq \nu,}
the following correspondence between memory addresses and
subscripted variables seems to be natural
\[
\begin{align*}
\text{BFS} & \leftrightarrow a[\mu] \\
\text{BFS}+1 & \leftrightarrow a[\mu+1] \\
& \vdots \\
\text{BFS}+\nu-\mu & \leftrightarrow a[\nu].
\end{align*}
\]
In general we have
\[(\ast) \text{ address } (a[i]) = \text{BFS}-\mu+i, \ \mu \leq i \leq \nu.\]
This is the storage mapping which associates with any sub-
scripted variable the corresponding physical address. \text{BFS}-\mu
is called the \textit{reduced initial address} \(a_{\text{red init}}\) \(a_{\text{red init}}\)
is the address of \(a[0]\) if the subscripted variable \(a[0]\) exists.
So, we have \[\text{address}(a[i]) = a_{\text{red init}}+i.\]
This formula allows an easier computation of address \(a[i]\)
than formula \((\ast)\), as \(a_{\text{red init}}\) is often known already at
translation time and at least known when the array declara-
tion is executed at run time.

Let us show how a simple assignment statement
\[x := y - a[5]\]
with \(\mu \leq 5 \leq \nu\) is to be translated into assembler language. We
assume that $a_{\text{red init}}$ is a fixed address computed at translation time so that the number $a_{\text{red init}}$ can be incorporated into the translated code. This is possible at least for static arrays.

$$
\begin{align*}
\text{EN}[a_{\text{red init}}] & \quad AC := a_{\text{red init}} \\
\text{STA } h & \quad h := AC \\
\text{EN}[5] & \quad AC := 5 \\
\text{ADA } h & \quad AC := AC + fi h \\
\text{STA } h & \quad h := AC \\
\text{LDA } y & \quad AC := y \\
\text{SBA, } I & \quad AC := AC - fi \text{ cont } h \\
\text{STA, } x & \quad x := AC \\
\end{align*}
$$

$\text{SBA, } I \text{ h resp. } AC := AC - fi \text{ cont } h$ is an instruction which we have not yet become acquainted with. The symbols $I$ resp. $\text{cont}$ indicate that the so called substitution bit of the operational part of the machine instruction SBA is to be 1 instead of 0.

<table>
<thead>
<tr>
<th>Operational part</th>
<th>Address part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

In that case not the content of cell $h$ but the content of the content (an address) of cell $h$ is subtracted from the content of the accumulator.

Another way to translate $x := y - a[5]$ is to use index registers which we assume that the machine has available some, e.g.

IR 1, IR 2, and IR 3.

If we know that at translation time IR 2 is free we may translate

$$
\begin{align*}
\text{EN}[5, 2] & \quad IR 2 := 5 \\
\text{LDA } y & \quad AC := y \\
\text{SBA, } a_{\text{red init}}, 2 & \quad AC := AC - fi A a_{\text{red init}} | IR 2 \\
\text{STA } x & \quad x := AC \\
\end{align*}
$$

The symbols $2$ resp. $| IR 2$ in the third instruction indicate that the so called index register bits of the machine
instruction SBA must be 10 instead of 00 (binary representation of 2 resp. 0).

\[
\begin{array}{c|c|c}
\text{index register bits} & \text{substitution bit} \\
\hline
1 & 0 & 0 \\
\end{array}
\]

operational part \hspace{1cm} address part

When the instruction is executed first, the address

\[ a = \text{a red init } + \text{ cont (IR 2)} \]

is determined by the instruction processing unit, afterwards the subtraction \( SBAa \) resp. \[ AC := AC - f_1 A \]
in the arithmetic unit is done.

We see that either the substitution property of a machine or the availability of index registers can be used to handle subscripted variables in machine code.

Two dimensional arrays can be stored conveniently in a row wise manner:

\[
\text{integer array } \quad a[u_2:v_2,u_1:v_1] \\
BFS \quad \leftrightarrow \quad a[u_2,u_1] \\
BFS+1 \quad \leftrightarrow \quad a[u_2,u_1+1] \\
\vdots \\
BFS+(v_2-u_2+1) \times (v_1-u_1+i)-1 \quad \leftrightarrow \quad a[v_2,v_1]
\]

Generally,

\[ \text{address}(a[i,j]) = \text{a red init } + K_1 \times i + j \]

where

\[ \text{a red init} = Df \quad BFS-K_1 \times u_2-u_1 \]

\text{a red init} is the address of } a[0,0] \text{ if this subscripted variable exists.}
If we should like to store three and more dimensional arrays
integer array \( a[u_r, v_r, \ldots, u_l, v_l] \)
row wise storing is to be generalized and we are led to a lexicographical linear ordering of subscripted variables. This leads to a storage mapping
\[
\text{address } a[j_r, \ldots, j_l] = a_{\text{red init}} + \phi_a(j_r, \ldots, j_l)
\]
where the reduced initial address is
\[
a_{\text{red init}} = \text{BFS-} \phi_a(u_r, \ldots, u_l) = \text{address}(a[0, \ldots, 0])
\]
the distance function is
\[
\phi_a(j_r, \ldots, j_l) = (\ldots((j_r \times K_{r-1} + j_{r-1}) \\
\times K_{r-2} + j_{r-2}) \ldots) \\
\times K_l + j_l,
\]
and the edge lengths are
\[
K_p = \text{Df } \nu_p - \nu_p + 1, \quad p = 1, \ldots, r
\]
At this stage we can see very clearly that it is advantageous to base addresses of subscripted variables on the address of \( a[0, \ldots, 0] \) and not on the address of \( a[1, \ldots, l] \), as it is done in some compilers.
\[
a_{\text{red init}} + \phi_a(j_r, \ldots, j_l)
\]
demands \( r-1 \) multiplications and \( r \) additions,
\[
\text{address}(a[1, \ldots, l]) + \phi_a(j_{r-1}, \ldots, j_{l-1})
\]
demands \( r-1 \) multiplications and \( 2r \) additions or subtractions. It is interesting to note that the edge length \( K_r \) is not necessary to compute subscripted variable addresses, \( K_r \) is only necessary to determine the new beginning of free storage after the array storage allocation.

Up to now we have used only number denotations as subscripts and we have used subscripted variables only as fixed address denotations of an array store, e.g. \( a[5] \). If \( i \) is a simple integer variable, then, \( a[i+5] \) is no longer a denotation for a fixed address of a storage place as the content of
i may vary. Actually, \( a[i+5] \) is a denotation for an address function. As \( a[i+5] \) may occur on the left hand of an assignment statement we must generalize our value assignments \( \tau_{af,f}^V \) where the first parameter is an address function \( af : S \rightarrow A \), the second parameter is a value function \( f : S \rightarrow V \). Let \( s : A \rightarrow V \) be a storage state. We have to define

\[
(s)\tau_{af,f}^V = s' | A \rightarrow V:
\]

\[
s' = Df^A
\]

\[
s'(x) = \begin{cases} s(x) & \text{if } x \notin af(s) \\ f(x) & \text{if } x = af(s) \end{cases}
\]

\( (s)\tau_{af,f}^V \) is undefined if \( af(s) \notin A \).

What address function does especially \( a[i+5] \) denote if \( i \) is a denotation for an integer variable?

\[
\phi(a[i+5]) = Df^\phi(a[s(\phi(i)) + 5])
\]

if \( s(\phi(i)) + 5 \in \{u:v\} \).

The assignment statement

\[
a[i+5] := \psi
\]

denotes the following storage state transformation

\[
(a[i+5] := \psi) = Df^\phi(a[i+5]) = \phi(\psi)
\]

where \( \tau_{af,f}^V \) is defined above.

11. RECURSIVE ADDRESS CALCULATION

It is an experience that a translated high level language program spends much of its run time with calculating addresses of subscripted variables. Therefore, it is necessary to think about optimization methods. Optimization is possible if subscripted variables occur in for statements. Let a for statement in ALGOL 60 have the form

\[
\text{for } i := a \text{ step } b \text{ until } c \text{ do } S_i
\]

where \( i \) is a simple integer variable, \( a, b, c \) are formulas, and \( S_i \) is a controlled statement. We deviate a
little from the prescriptions of the ALGOL 60 Report and translate the for statement in the following way

\[
i := a_i; \\
GVb_i := b_i; \\
GVC_i := c_i; \\
goto TEST_i; \\
\text{BACK}_i : i := i + GVb_i; \\
\text{TEST}_i : \text{if } (i - GVC_i) \times \text{sign}(GVb_i) > 0 \text{ then goto END}_i; \\
S_i; \\
goto \text{BACK}_i; \\
\text{END}_i;
\]

\(GVb_i\) and \(GVC_i\) are generated integer variables. If \(b_i\) or \(c_i\) are simple variables or constants, \(GVb_i\) and \(b_i\) resp. \(GVC_i\) and \(c_i\) may be identified.

Now, let a subscripted variable

\[a[s_r, \ldots, s_l]\]

occur in the controlled statement \(S_i\):

\[
\text{for } i := a_i \text{ step } b_i \text{ until } c_i \text{ do} \\
\ldots + a[s_r, \ldots, s_l] + \ldots.
\]

The expression

\[\ldots + a[s_r, \ldots, s_l] + \ldots\]

must be translated in the following way:

\[GVa := a_{\text{red init}} + \phi_a(s_r, \ldots, s_l); \]

\[\ldots + \text{cont } GVa + \ldots\]

\(GVa\) is a generated variable associated with the subscripted variable \(a[s_r, \ldots, s_l]\).

Furthermore, let \(s_{\rho}, \rho = 1, \ldots, r\) be linear functions of \(i\)

\[s_{\rho} = t_{\rho} + ixu_{\rho}\]

where \(t_{\rho}, u_{\rho}\) are independent of \(i\). Then

\[a_{\text{red init}} + \phi_a(t_r, \ldots, t_l) + ix\phi_a(u_r, \ldots, u_l)\]
as the distance function is a linear function. At run time we see that for the starting value of $i a[s_r, \ldots, s_i]$ has the following initial address

$$a_{\text{red init}} + \phi_a(t_r, \ldots, t_i) + i \times \phi_a(u_r, \ldots, u_i).$$

In every step from $i$ to $i+GVB_i$ the address of $a[s_r, \ldots, s_i]$ increases by the constant value

$$GVB_i \times \phi_a(u_r, \ldots, u_i).$$

Under these circumstances we may extract

$$GVA := a_{\text{red init}} + \phi_a(s_r, \ldots, s_i);$$

out of the controlled statement:

$$i := a_i;$$
$$GVB_i := b_i;$$
$$GVC_i := c_i;$$
$$GVu := \phi_a(u_r, \ldots, u_i);$$
$$GVA := a_{\text{red init}} + \phi_a(t_r, \ldots, t_i) + i \times GVu;$$
$$GVA_i := GVB_i + GVu;$$
$$\text{goto TEST}_i;$$
$$\text{BACK}_i: i := i + GVB_i;$$
$$GVA := GVA + GVA_i;$$
$$\text{TEST}_i: \text{if } (i-GVC_i) \times \text{sign}(GVB_i) > 0 \text{ then goto END}_i;$$
$$\ldots + \text{cont GVA + } \ldots;$$
$$\text{goto BACK}_i;$$

END_i:

GVu and GVA_i are additional generated variables.

We apply this method of recursive address calculation for subscripted variables to an example dealing with partial differential equations:

$$\text{for } k := a_k \text{ step 1 until } c_k \text{ do}$$
$$\text{for } k := a_i \text{ step 1 until } c_i \text{ do}$$
$$\ldots (A \times a[k, i] + a[k, i+1] + a[k, i-1]$$
$$\ldots a[k+1, i] + a[k-1, i])/8 \ldots$$

$a_k$, $c_k$, $a_i$, $c_i$ are assumed to be variables.
Application to the inner loop gives:

\[
\text{for } k := a_k \text{ step } 1 \text{ until } c_k \text{ do }
\begin{align*}
\text{begin} \\
& i := a_i; \\
& GV_{a_1} := \text{red init } + \phi_a (k, 0) + i; \\
& GV_{a_2} := \text{red init } + \phi_a (k, 1) + i; \\
& GV_{a_3} := \text{red init } + \phi_a (k, -1) + i; \\
& GV_{a_4} := \text{red init } + \phi_a (k + 1, 1) + i; \\
& GV_{a_5} := \text{red init } + \phi_a (k - 1, 1) + i; \\
& \text{goto TEST}_i; \\
\text{BACK}_i: & \quad i := i + 1; \\
& [GV_{a_1} := GV_{a_1} + 1; \\
& \quad \vdots \\
& \quad GV_{a_5} := GV_{a_5} + 1;] \\
\text{TEST}_i: & \quad \text{if } (i - c_i) > 0 \text{ then goto END}_i; \\
& \quad \ldots (4 \times \text{cont } GV_{a_1} + \text{cont } GV_{a_2} + \text{cont } GV_{a_3} \\
& \quad + \text{cont } GV_{a_4} + \text{cont } GV_{a_5}) / 8 \ldots ; \\
& \quad \text{goto BACK}_i; \\
\text{END}_i: & \quad \text{end}
\end{align*}
\]

We have used the facts that GV = \(a \phi (0, i)\) is 1 for all subscripted variables, that \(b_i = i\), and that GV\(\Delta_j\) is 1. These facts can be recognized by a compiler at translation time.

As the five formulas

\[
\text{are linear in } k, \text{ their values can be calculated recursively, too:}
\]

\[
\text{are linear in } k, \text{ their values can be calculated recursively, too:}
\]

\[
\text{are linear in } k, \text{ their values can be calculated recursively, too:}
\]
k := a_k;
GVf_1 := a \text{red init} + k \times K_1;
GVf_2 := a \text{red init} + 1 + k \times K_1;
GVf_3 := a \text{red init} - 1 + k \times K_1;
GVf_4 := a \text{red init} + K_1 + 1 + k \times K_1;
GVf_5 := a \text{red init} - K_1 + 1 + k \times K_1;
goto \text{TEST}_k;
\text{BACK}_k : k := k + 1;
GVf_1 := GVf_1 + K_1;
: ;
GVf_5 := GVf_5 + K_1;
\text{TEST}_k : \text{if } (k - c_k) > 0 \text{ then goto END}_k;
\begin{aligned}
i := a_i; \\
GVA_1 := GVf_1 + i; \\
: ; \\
GVA_5 := GVf_5 + i; \\
goto \text{TEST}_i; \\
\end{aligned}
\text{BACK}_i : i := i + 1;
\begin{aligned}
GVA_1 := GVA_1 + 1; \\
: ; \\
GVA_5 := GVA_5 + 1; \\
\end{aligned}
\text{TEST}_i : \text{if } (i - c_i) > 0 \text{ then goto END}_i;
\begin{aligned}
&\ldots (4 \times \text{cont } GVA_1 + \text{cont } GVA_2 + \text{cont } GVA_3 \\
&+ \text{cont } GVA_4 + \text{cont } GVA_5)/8 \ldots ;
\end{aligned}
goto \text{BACK}_i;
\text{END}_i : \text{end};
goto \text{BACK}_k;
\text{END}_k :

Here we have used the facts that GVu=\phi_a(1,0) is K_1 for all five formulas and that b_k is 1. These facts can be figured out by a compiler at translation time, too.
It is well known to assembler code programmers that matrix tasks can be done advantageously by the help of index registers. Index registers can be put into work even by a compiler in an automatic way. In the last two programs we see that the variables GVA₁, ..., GVA₅, the values of which are addresses of subscripted variables, are raised by the same value (here 1). These five subscripted variables are now associated with the same index register, say IR 2, which must be free at run time. We replace the framed pieces of program by the following pieces of program:

\[
\begin{align*}
\text{IR2} & := 0; \\
\text{GVA}_1 & := \text{a}_{\text{red init}} + \phi_a (k, 0) + i | \text{IR2}; \\
\vdots & \\
\text{GVA}_5 & := \text{a}_{\text{red init}} + \phi_a (k-1, 1) + i | \text{IR2};
\end{align*}
\]

respectively

\[
\begin{align*}
\text{IR2} & := 0; \\
\text{GVA}_1 & := \text{GVf}_1 + i | \text{IR2}; \\
\vdots & \\
\text{GVA}_5 & := \text{GVf}_5 + i | \text{IR2};
\end{align*}
\]

and

\[
\text{IR2} := \text{IR2} + 1;
\]

The operation \(|\text{IR2}\) means that the index register bits of the cells GVA₁, ..., GVA₅ shall be occupied by 10, the binary representation for 2. We exploit the fact that an average computing machine executes index register modification first and substitution afterwards. Here we should state that for most for statements there is no advantage in taking an index register for the loop variable. Index registers should be associated with subscripted variables in situations as shown above.

It is obvious that a hand programmer cannot do his job much better than a compiler, which has these recursive address calculation features. Recursive address calculation can
be done in ALGOL as well as in FORTRAN.

It is clear that the compiler has to do some extra work:

1. The compiler must ensure that the value of the loop variable is changed only by the for clause.

2. The compiler must ensure that the expressions $t_\rho^*$ and $u_\rho^*$ are independent of the loop variable and of variables which may change their values during the execution of the for statement.

3. The array declaration must be outside of the for statement.

Recursive address calculation decreases the necessary computing time of a program quite considerably. This is especially true for programs which deal with numerical and differential equation problems.

LITERATURE


LIST OF PARTICIPANTS

1972 CERN COMPUTING AND DATA PROCESSING SCHOOL, PERTISAU, AUSTRIA

Lecturers

CONETTI, S. CERN, Geneva, Switzerland
JAMES, F. CERN, Geneva, Switzerland
KEYSER, R. CERN, Geneva, Switzerland
OBERSKI, J.E.J. Instituut voor Kernfysisch Onderzoek, Amsterdam, Netherlands
OLLE, T.W. Independent Consultant, Branksome, Poole, Dorset, UK
PERRY, M.A. European Space Research Organisation, Noordwijk, Netherlands
PYLE, I.C. AERE, Harwell, Berkshire, UK
RUSSELL, R.D. CERN, Geneva, Switzerland
WIND, H. CERN, Geneva, Switzerland
ZACHAROV, B. Daresbury Nuclear Physics Laboratory, Daresbury, Lancashire, UK, and CERN, Geneva, Switzerland

* * *

ADAMS, J. CERN, Geneva, Switzerland
AYED, R. Centre d’Etudes Nucléaires, Saclay, France
BARTA, G. CERN, Geneva, Switzerland
BARTNIK, E.A. Institute of Theoretical Physics, The University, Warsaw, Poland
BELL, D.A. Department of Computer Science, National Physical Laboratory, Teddington, Middlesex, UK
BONTHONNEAU, F. Laboratoire de Physique Corpusculaire, Caen, France
BROWN, J. Atlas Computer Laboratory, Chilton, Didcot, Berkshire, UK
BRUNDIERS, H.-J. Fakultät für Physik der Universität, Freiburg, Fed. Rep. Germany
CALINET, A. Laboratoire de Physique Nucléaire et de Haute Énergie, Paris, France
CARLSON, B. Research Institute of the Swedish National Defence, Stockholm, Sweden
CASAL ZAMORANO, E. Instituto de Física Corpuscular, Facultad de Ciencias, Valencia, Spain
CHAFFMAN, W.S. Rutherford Laboratory, Chilton, Didcot, Berkshire, UK
COFFARO, P. Laboratorio di Fisica Coassiale e Tecnologie Relative, Palermo, Italy
CRLJNS, F. Physikos Laboratory, Toenienveld, The University, Nijmegen, Netherlands
DAVIES, M.P.H. Department of Nuclear Physics, The University, Oxford, UK
DE GIORGI, M. Istituto di Fisica dell’Università, Padua, Italy
DOOLE, D.J. School of Mathematics, The University, Bristol, UK
DUVIVIER, M.R.M. Centre de Calcul de Physique Nucléaire, Faculté des Sciences, Paris, France
ELCOMBE P.A. Cavendish Laboratory, Cambridge, UK
EVANGELIDES, E. High Energy Physics Department, The University, Manchester, UK
FAYE, A. Laboratoire de Physique Atomique, Collège de France, Paris, France
FRANKICK, V. CERN, Geneva, Switzerland
GOSMAN, D. Zeeman Laboratorium, Amsterdam, Netherlands
GREENLEAF, J.R. AEGER, Harwell, Didcot, Berkshire, UK
GRIGORJEW, E. Institute for Theoretical and Experimental Physics, Moscow, USSR
GRITZALI-PAPADOPOULOU, F. N.R.C. "Democritos", Athens, Greece
GUYONNET, J.-L. A. Laboratoire de Physique Corpusculaire, Centre de Recherches Nucléaires, Strasbourg, France
HARING, G. Institut für Angewandte Mathematik und Informationsverarbeitung, Technische Hochschule, Graz, Austria
HOPSON, B. CERN, Geneva, Switzerland
HOULDEN, M.A. Department of Physics, The University, Liverpool, UK
HULTH, P.O. Institute of Physics, Stockholm, Sweden
JENNI, P. Institut für exakte Wissenschaften, Abt. Hochenergiephysik, Bern, Switzerland
JENNINGS, D.M. System Dynamics, Dublin, Eire
KUTIN, S. Institute of High Energy Physics, Serpukhov, USSR
LATHYI, L.-B. Centre d'Informatique, Ecole Royale Militaire, Brussels, Belgium
LENTS, B.E. International Atomic Energy Agency, Vienna, Austria
LODONGIN, D. Joint Institute for Nuclear Research, Moscow, USSR
LORETTI, M. Istituto di Fisica dell'Università, Padua, Italy
MAJDAN, D. Rutherford Laboratory, Chilton, Didcot, Berkshire, UK
MANOUSSIS, E. Centre de Calcul de Physique Nucléaire, Paris, France
MAYER, A. The University, Tel-Aviv, Israel
MEYER, J.-M. Centre de Recherches Nucléaires, Strasbourg, France
MIRABELLI, G. Istituto di Fisica dell'Università, Rome, Italy
MITAROFF, W.-A. Institut für Hochenergiephysik, Vienna, Austria
MORTENSEN, B.N. Niels Bohr Institute, Copenhagen, Denmark
MOULDALE, N.R. Department of Physics, The University, Birmingham, UK
NUMMI, P. Department of Nuclear Physics, The University, Helsinki, Finland
PALAZZI, P. CERN, Geneva, Switzerland
PHILIPPE, R. Laboratoire de l'Accélérateur Linéaire, Université Paris XI, Orsay, France
PIESS, H. Schweizerisches Institut für Kernforschung, Villigen, Switzerland
RADIG, B. Physikalisches Institut der Universität, Bonn, Fed. Rep. Germany
RENTON, P.B. Vrije Universiteit Brussel, Dienst Elementaire Deeltjes, Brussels, Belgium
RICHMOND, P.M. Daresbury Nuclear Physics Laboratory, Warrington, Lancashire, UK
SALA, S. Istituto Nazionale di Fisica Nucleare, Milan, Italy
SIEBERT, M.F. Zentrum für Rechentechnik, Deutsche Akademie der Wissenschaften, Berlin, Germany
SILVERMAN, A. CERN, Geneva, Switzerland
STUDACH, T. Eidgenössisches Institut für Reaktorforschung, Würenlingen, Switzerland
TIRINGER, M. J. Stefan Institut, The University, Ljubljana, Yugoslavia
VENTURI, D. Centro Nazionale Analisi Fotogrammi, Istituto Nazionale di Fisica Nucleare, Bologna, Italy
VERVERS, F. Mathematical Department, Delft University of Technology, Delft, Netherlands
ZITO, G. Istituto di Fisica dell'Università, Bari, Italy