Accounting for the soldier's pay—Organization of programming

By Major W. S. Caskey, M.B.E.

This paper discusses the organization of programming within a large-scale data-processing installation, and comments on some of the problems which are vitally important when deciding the overall programming concept. The paper relates primarily to the Royal Army Pay Corps installation, which is described in the paper by Lt.-Col. D. W. Moore, which begins on p. 249.

Statistical data

Statistics are a dangerous basis on which to commence my talk, but perhaps the basic problem of programming this application has been the sheer size and complexity of it. One has to imagine a daily processing which requires three main programming runs—an edit run of some 6,000 instructions, an updating run of some 35,000 instructions, and an output distribution run of some 15,000 instructions. The monthly processing involves three major runs varying between 5,000 and 14,000 instructions. In our daily processing of soldiers' accounts there can be any number, in any permutation, of approximately 400 possible changes. The great majority of these involve considerable validity checking. Many are retrospective to action already taken, since we maintain a historic record for up to a year. All, whether retrospective or not, or in whatever combination, are automatically processed. The application produces up to 70 varying forms. Some of these have varying formats.

The master file record for each soldier varies from about 400 to 1,900 digits; the average length is 800. The record itself is divided into two—a fixed portion which contains data common to all soldiers, and a variable portion containing data which relate only to soldiers in certain circumstances (married details, allotment particulars, etc.).

Method of approach

How then should one set about a programming task of this magnitude? Well, I will outline our views, but I wish to make it clear that some of these will not coincide with the practice we actually employed; but where this is the case we would have been better advised to follow the course put forward.

By-passing completely the feasibility-study stage, I think the first requirement is to have a clear definition of the overall task. Then, I think, the controlling programmer must project his thought to visualizing the flow of work around the computer and its supporting units—in short, the computer system. He must gear his programming concept not only towards an efficient program but, of far more importance, to one which is flexible, is capable of amendment easily, and meets the needs of a good steady flow of work.

While he is doing this, the programmers proper (or the systems analysts, as the case may be) can draw up a precise and critical assessment of what output is required from the computer system. In short, one builds up a programming objective. This evaluation must include a definition of the precise contents of each form, even for those forms used infrequently. The number of forms obtained may well be a critical factor in the nature of the processing runs to be adopted.

The next stage is a critical examination of the input data currently available, and an assessment of any further data of value which might be obtained by reasonable change. When this has been completed the master record can be drawn up containing those historical facts and current accumulations, etc., necessary to provide the output already defined. One can then consider, for example, whether the record should be fixed or variable in length. In our case, we chose a record which contains both a fixed and, where appropriate, a variable portion.

When these stages have been completed, one can determine the means by which the input data can be processed against the master record to ensure that all regulation criteria are met, and that the necessary output is provided—in short, one can begin to build up the logic of the programming runs.

Of course, as we progressed, these three requirements became somewhat interrelated. At quite an early stage we were able to define the basic requirement of how to establish accurately the soldier's entitlement at any point in time. We were able to decide to omit rates of pay from the actual record almost entirely (holding centrally large general tables). We decided to hold details of the soldier's status related to time—all changes simply being added to the record in 'skeleton' form in correct chronological order.

Any casualty received is converted into a 'skeleton' which is placed in the soldier's magnetic-tape record in its correct chronological position. Any 'skeletons' already on the record relating to a date later than this currently received casualty are taken into account in assessing the effect of the casualty and determining the current rate of pay to be notified to the unit. This basic concept has never changed and meets all the requirements of retrospectivity—a common feature of our processing.

Then, of course, came the detailed break-down of work; first of all to programming runs within the programming master concept and, secondly, within runs, to subroutines and special machine program requirements.

Programming control criteria

Before I deal with the organization we used to effect this task I feel I ought to list certain important points which we had to consider before being definitive on this aspect. They might even be called basic programming control principles, applying generally to large-scale data processing. These are as follows:

(*a*) To be repetitive, the computer system had to be visualized at the earliest possible minute. Very often there are certain factors which are fundamental and which must be adhered to regardless of circumstances. For example, in our case we receive input data from 15 offices all over the country. They can arrive by either of two postal deliveries, and in an order which varies from day to day. This factor means that our editing runs must be processed at least twice on any given day and, more fundamentally, that the master file must be so contained that a variable sequence of offices present no problems. The number of output forms, in our case about 70 built up initially in random order as a result of individual processing requirements, demand a further run—a distribution run. This run builds up the forms in proper format, gives the facility of carrying out a great deal of additional processing where required without putting additional load on the main updating run, and puts all like forms in a predetermined sequence on to print tapes. You will note that such a concept significantly changes the logic of the main updating run if previously this has been viewed solely as a programming task. In short, the nature of the programming runs can be seen clearly from an examination of the system requirements considered in the light of machine configuration limitations.

(*b*) All input data had to be examined by the computer itself to check that the manual processes have been completed correctly. In my view this is a cardinal principle, to be adhered to at all costs, both for initial conversion of master records and for daily updating. We were extremely strict in this field—yet we still had many cases slipping through the net. This editing can be done either in part on electrical accounting machinery, or on a pre-main updating process on the computer, or as part of the updating process on the computer. Primarily, we chose the two former, doing the bulk of our editing before any run which contains a master file. Certain advantages accrued, among which were:

(i) accuracy of data before having the master file on the computer;
(ii) computer-agreed control totals produced quickly (and again before processing the master file);
(iii) facility to build up interrelationship of casualties for soldiers;
(iv) increased flexibility (more tape units available, etc.).

(*c*) Every programmer had to be made to realize that his work would be run eventually on the computer in his absence, possibly at 2 o'clock in the morning by a console operator without his special knowledge. Therefore he must have the console operator's needs in his mind at all times. There must be no legs of his program which are labelled "This cannot happen." They *will* occur together with many more he never even thought of. Therefore, corrective transfer action for every possible eventuality must be recorded in a form which is readily digestible.

(*d*) A good detailed brief for programmers had to be issued at the earliest possible stage. Such a brief must lay down programming conventions and instructions. It should also define precisely the documentary standard required for flow charts, program folders, etc. etc. Ideally it should also define the best means of desk checking.

(*e*) Good and standard documentation had to be insisted upon. Somehow, as the organization builds up, there must be the means of allotting storage space, co-ordinating programmers' work, allocating universal or common working stores, constants and subroutines. Carrying this thought even further forward to operational running, there must be the means of patching programs speedily and effectively. We are exceptional in this instance (we have had something like 700 patched instructions in our programs, plus two re-aligning scrutinies arising from the recent pay review, and we have about 70 revised instructions on average each week) —nevertheless the principle remains common to most installations.

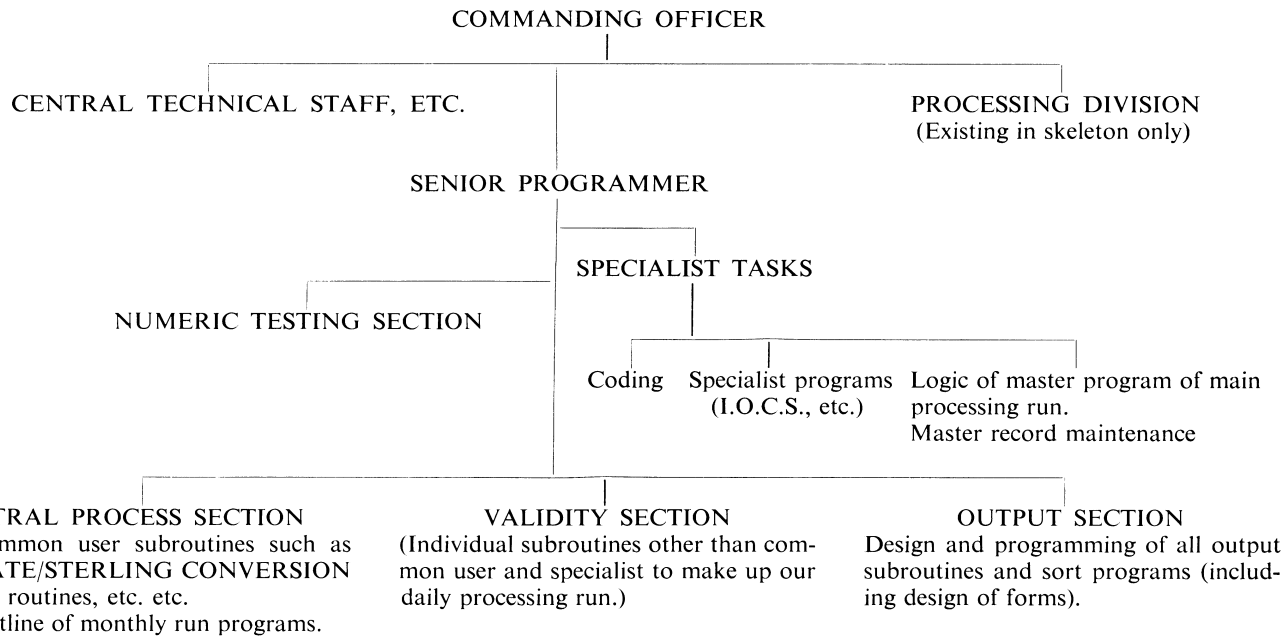
(*f*) The problem of coding had to be given early priority. In our view this is a wide issue. In our own case we believe that the basic codes for (i) input data prepared manually in the outlying offices for processing on the computer against the soldier's master file tape record, for (ii) programming purposes, and for (iii) interpretation of output coding, must be basically the same. We went to endless trouble to do this, and now a clerk can become a programmer and then go out to a unit and use the same basic coding knowledge or extensions of it throughout. We use three codes. The first is the Card/Casualty Code which is the controlling medium for the electrical accounting machines and for the computer. The second is the Error Rejection Code. The third is a General Purpose Code, used for input and output definition of data.

(*g*) May I here inject a personal view when speaking of large-scale programs. The whole programming concept should be built in sections which are suitable for easy withdrawal and replacement. Complex programming, a joy in small programs, can hold hidden dangers in large-scale work, particularly where changes are frequent and where the degree of integration is great.

CHART A

Table 1

R.A.P.C. Electronic Accounting Development Unit—Outline Organization of Program Division, July 1959



(*h*) Where there was one exceptionally large programming run we found it advisable to set aside one programmer to write the master program logic. This practice saved many months of argument over which subroutines should call in others, etc. etc. I record this point with feeling as we suffered a little initially on this score.

Programming staff organization

And so to the programming organization we used from July 1959 (see Table 1). Prior to this period we had a programming team which worked on the feasibility study, etc., and did much of the subroutine logic. For example, this team devised the means whereby we store data related to time, and the means whereby we record status and not pay rates (finding these from central pay tables when required). This logic was built up in isolation before all the various processing runs were determined, but it has stood the test of time.

From July 1959 we had a Senior Programmer who received his instructions from the Commanding Officer. He kept, under his own direct control, programmers dealing with the definition of the master record, specialist programs, and coding. Thus nothing new could be injected or any sizeable change made without his being fully in the picture. This is in itself a problem in large-scale programming when there is a large number of programmers all busily creating new proposals!

We started off with three basic sections. The first prepared the common user subroutines, the second prepared the large number of validity routines to meet the regulations relating to the 400 types of variation in entitlement which can occur, and the third designed the output forms and prepared the print routines.

These programmers prepared logical flow charts for their tasks. These charts were examined initially by a numeric test section. This section had the basic function of confirming the logic used within subroutines, and of correcting errors. However, it also provided the means of assessing considerable statistical data. The sizes and frequency of the use of subroutines became clearly defined. Many commonly used groups of instructions, working stores, constants, etc., were extracted and made into what we call universal areas. These universal factors are dealt with by the person responsible for the master program, and are made available to all programmers in the fastest storage medium which the frequency of their being required justified. This process took rather a long time, and reduced the overall size of the runs considerably. However, four valuable additional bonuses occurred. Firstly, this section rapidly provided a nucleus of programmers with a wide view of the programming concept. Secondly, the means whereby a sound storage allocation could be made was provided—and I would like to record here that the basis on which this was done has not been changed. Thirdly, the continuous training of programmers was facilitated, as errors were referred back to the offending programmer, and common errors were circulated in programming directives, etc. Fourthly, the best means of machine testing each subroutine was examined and commented upon within this section.

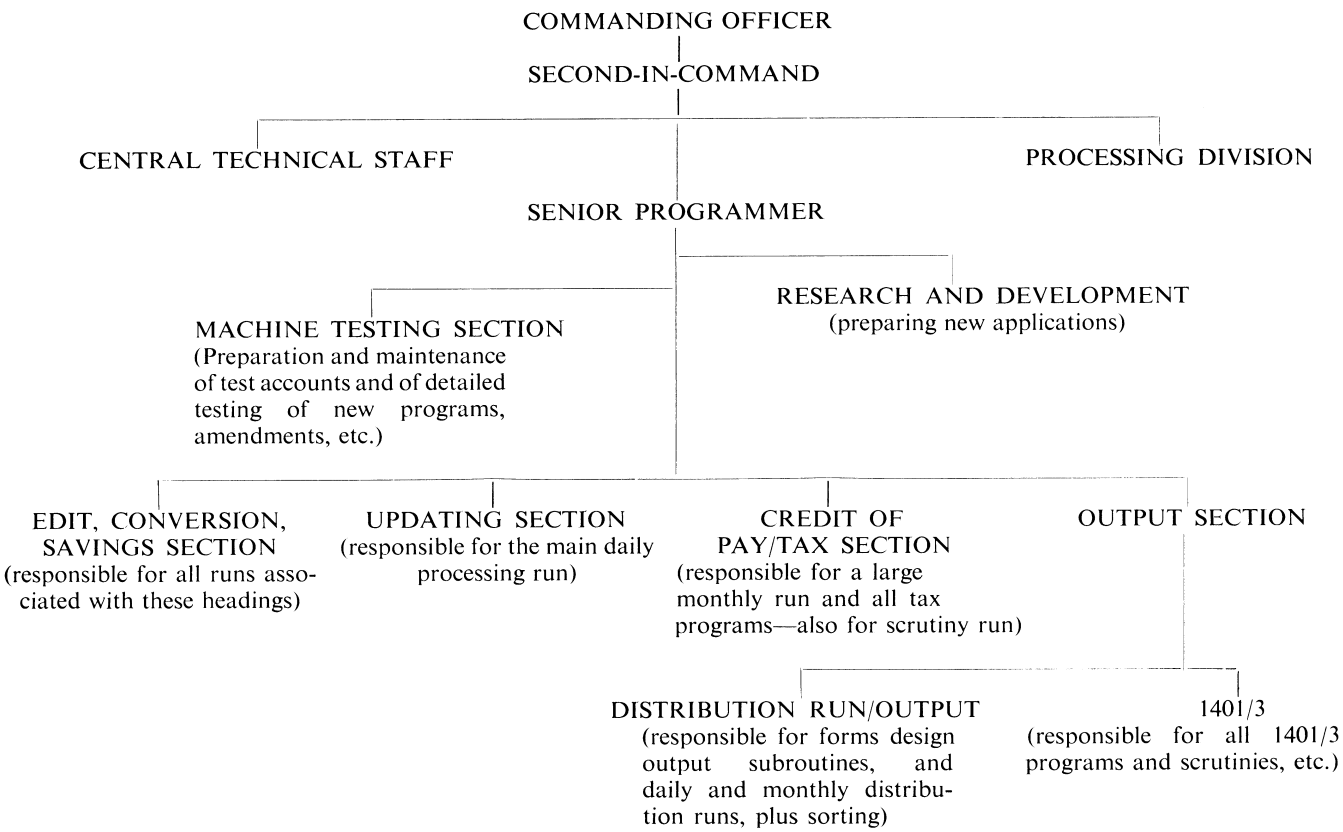
This section was the main means of detailed co-ordination of final effort. When the logic was cleared,

CHART B

Table 2

R.A.P.C. Electronic Accounting Development Unit—Outline Organization of Program Division, March 1962

Unit became Operational 1 *November* 1960



the initiating programmer prepared detailed "block diagrams" (i.e. in our language, logical charts with actual compiler instructions within the blocks).

These in turn were critically examined by the numeric testing section, who put detailed numeric examples through every single instruction, sending the work back again and again until a satisfactory standard was reached.

The specialist programmers I referred to earlier studied the problems of tape labelling, of the input/output control system, and of correction routines, etc. During the early stages we used a relatively senior programmer to prepare a very large general-purpose code, a sizeable card and casualty code, and an error rejection code.

Overall control of all this activity was exercised in two ways: first, centrally within the unit by means of Planning Instructions which clearly defined requirements, rulings, and the principles to be followed; secondly, within the programming set-up by detailed Programming Instructions which defined precisely programming techniques, the ever-changing library of common subroutines, amendments/additions to codes, alterations to master records, etc. etc. I cannot over-stress the need for a sound co-ordination medium. We were extremely careful, and yet there were far too many instances of different interpretation of instructions and of meetings, etc. Each of these eventually wasted valuable machine time.

Changes in programming organization

The programming organization should be developed continuously—if it remained static then I would fear that proper control was not being exercised. Let us now look at Table 2. In our case, the numeric test section disappeared and a machine testing section grew as we approached operational running. Indeed, while going to press I should note that the numeric test section has returned and is saving valuable machine time by reducing the number of errors which would have turned up during test periods on the computer.

Gradually, as we gained machine experience, testing procedures were developed, better means of correcting assembly errors were found, and a strict control of

'patches' to subroutines, etc., after testing was devised. The absolutely rigid control of this field is a tremendous problem in itself, and no laxity can be allowed. The problems of the effect of changes, both within a programming run and between programming runs, are major ones, and clear-sighted control is imperative.

After the initial few months of operational running, the responsibility of individual programmers was raised to runs instead of subroutines within a run. Currently, the examination of further possible applications is becoming more detailed as some of the experienced programmers are freed from the initial task. Optimization of programs, as and when the opportunity or the need arises, has been and is still a profitable exercise. New languages and programming aids are being reviewed continually. Within the programming organization we have a few non-programmers who maintain test accounts and devise test data for re-written patched routines. They critically examine all the output resulting from testing sessions—in fact they form an integral part of the testing section, and the programming organization.

Concluding remarks

Each of the points mentioned above, for example Coding or Storage Allocation or Machine Testing, is worthy of an hour's talk in itself. I can only hope to have outlined some of the problems and sketched out a possible means of approach.

Book Reviews

Data Acquisition and Processing in Biology and Medicine, Edited by KURT ENSLEIN, 1962; 191 pp. (Oxford: *Pergamon Press*, 50s.)

I was recently approached by the editor of a medical research journal to referee a paper submitted by an author from the United States who was proposing to make a quantitative science of a particular branch of medicine—no British doctor could be found to pass judgement on the paper. As an applied mathematician I was intrigued to find that the non-medical references in the paper were to geological papers. This reflects an attitude of mind on the part of U.S. research workers which is wholly to be applauded and which, outside operational-research circles perhaps, is unfortunately too seldom found in Britain. The explanation, at least in part, of this greater U.S. willingness to cross inter-disciplinary barriers is to be found in the impact which electronic instrumentation in general and computers in particular have had in the U.S. There, good computing facilities in universities and research centres are the norm; here, they are still the exception.

An example of what this means in scientific endeavour is to be found in the book under review, which is the edited proceedings of the 1961 Rochester Conference. The reports cover papers read at five sessions, concerned with computers in biology and medicine, computers and psychiatry, pattern recognition, clinical and research instrumentation for biological systems, and instrumentation for electrocardiography and electroencephalogy. It is interesting to note the degree of automation aimed at—one writer reports that electrocardiograms are already recorded on tape for running into a computer, but it is planned to use the cardiograph as an on-line device to the computer. The combination of intricate medical instrumentation, advanced computer techniques and highly sophisticated statistical treatment in some of the experiments is fascinating.

One of the papers on pattern recognition extends to the study of chest X-ray photographs the techniques used to assess the cloud photographs taken from the Tiros satellite.

The papers on computers have much to interest British readers. To me, most impact was made by the remarks by Lusted on the problems of education facing medical schools. Doctors, he says, must somehow be taught to compute when necessary, and he cites the work of doctors at Tulane University who have developed a computer program called "Probe" which allows the medical researcher who does not know programming to run his own medical data on the computer; they attach great importance to allowing the researcher "to get his hands on the machine."

I am not, of course, competent to review the papers in this volume for their medical contents; but I can wholeheartedly commend the book to British research workers if only for the insight it gives us of the attitude and approach of our transatlantic colleagues to research problems.

Until British universities and research establishments are equipped with computers in depth, as are their U.S. counterparts, one cannot imagine such a volume being written by British research workers.

ANDREW YOUNG.

Modulation and Coding in Information Systems, by GORDON M. RUSSELL, 1962; 260 pp. (London: *Prentice-Hall International Inc.*, 42s.)

The purpose of this book, as stated in the Preface, is "to give an introduction to the theory of information processes, primarily those of modulation and coding . . . applicable to the fields of power-system control, industrial control, data transmission and processing and all types of communication . . .". In saying this, the author does the book a slight disservice, for he suggests that the material is theoretical, dealing perhaps with information theory, coding theory and the like. But there are several excellent texts already on the market dealing with such theoretical aspects of communication and control systems. By contrast, there is a plethora of