This paper of December 1958 was circulated privately at RAND, SDC, and at SAC Headquarters. In the summer of '58 I was transferred as RAND liaison to SAC in Omaha for a year, to try to forestall what seemed inevitable: 465L (Strategic Air Command Control System) would be a multi-million dollar boondoggle, largely due to loosy software techniques. I helped not at all - the fiasco occurred on schedule - but I did try to beat the drum once again for standardization/restriction/structuring.

On page 9 I get down to basics, and even go so far as to hope that the hardware will be cooperative. (Once again, machine language was the order of the day). One topic is lightly touched on here, but then sort of swept under the table: This is the problem which I use the label "CONTEXT" to refer to, and which includes both spatial (what data does routine refer to) and temporal (when does the reference apply) aspects. Push-down storage isn't always the answer, although vital for recursion processes. [Heirarchical context may also be important - the processes of a routine may be dependent upon the level in the heirarchy at which it is currently being executed; this is vital in complex information processing, e.g., A.I or C.S]

On the last page I set down a few goals & I think it is this page which Fred Gruenberger remembers.
The paper was largely ignored.

Introduction

This discussion is a first attempt to set down some ideas relating to the problems of programming a large scale computer which is part of a large scale control system. Most of the remarks apply in general to any such system; others are specifically directed towards SACCS.

The fundamental objective here is to assure that the most advanced state of the art techniques of programming are applied, just as a requirement has already been made for the best computer hardware available at this time. Also, it is hoped that some of these ideas will assist the designers of the computer order code to help realize these techniques. It is also important to avoid wasted effort which would result if programs were produced before sufficient thought had been given to their design and integration into the over-all system. Such programs often result in inefficient operation later on and can even be completely incompatible with the final operating program. It is necessary, in a large computer programming system such as SACCS, to not merely "code the program," but rather to design a programming system, which will provide control over the program as it is written to insure that the desired results are attained, to provide documentation of the resulting program, and to provide documentation of the system. The remarks which follow are directed toward providing a starting point for such programming system design.

An apology should be made at this point on two counts. First, many topics discussed here may sound like trivial details;

these however, we feel to be fundamental, and their lack of application may indeed make a great deal of difference in the final system operation. Second, an analogy approach will sometimes be used, as many programming words and concepts are unfamiliar, undefined, or ambiguous.

Programming System Organization

In discussing the organization of the programming system, it is first necessary to define the difference between a program and a programming system. A program may be written as a complete entity, with no knowledge of the nature of any other programs which may be written for the same computer. Its only objective is that of producing the final answers in the time allotted. A programming system, on the other hand, provides a framework in which to construct a great number of programs, all of which will eventually be required to work together, at the same time, to produce the final results. It is unnecessary and extremely undesirable for one programmer to know all the intimate details of the code another programmer is writing; it is very necessary, on the other hand, for him to know the requirements and characteristics that each must have in order to be able to integrate into the final product. An analogy here would be in the design of the computer itself. A fair amount of the effort in computer design is devoted to insuring that, for instance, the designers of the tape system need not be familiar with the operation of the arithmetic element of the machine, but still operate together properly. At the same time, any redundant effort should be eliminated in the design of the basic circuit building blocks.

The objectives of the programming system design then are as follows.

1) To provide an orderly framework within which to build the required programs.
2) To make programs easier to write and easier to check out.
3) To provide a means for integrating various programs on a time shared basis in a real time application.
4) To delegate to the machine as many programming and machine operating tasks as feasible.
5) To develop standards of programming to keep the over-all continuity of the project independent of the personnel involved.
6) To provide standards for documentation and communication among programs.
7) To provide a means whereby the programming tasks may be delegated to a large number of programmers while insuring compatability at all stages.
8) To insure that the operating program complex can be easily updated as system requirements change.

The approach to this organization is threefold:

1) An over-all operational philosophy of the programming system will be defined, which will be tailored to meet the present and anticipated needs of the entire control system.

2) Specifications for the subsystem program components will be outlined. These components will be integrable, on a time shared basis, to comprise the complete operating program.
3) Standard methods of organizing the basic computer instructions into hierarchies of successively greater complex information processing capability, will be defined. These hierarchies will eventually satisfy the requirements of 1 and 2.

In the language of computer design, these correspond to 1) The over-all system characteristics which the system must have to do the proposed computing job; 2) the subsystems required to make up the required system (e.g., tapes, drums, input, output equipment, arithmetic element, etc.); 3) the electronic design techniques which will be used, the basic circuit module design, and the production methods to be employed.

Over-all Operational Philosophy

The nature of the task required of the computer, i.e., it must be capable of providing control information at any instant of time, dictates a mode of operation that is time shared with the other functions which the computer must provide. These include program checkout, planning programs (trajectories, war gaming, etc.), computer preventative maintenance and diagnostic programs, system training and exercising programs, etc. It is almost certain that a human operator or operators would be incapable of controlling the machine in its execution of these various functions either rapidly enough or with a sufficient degree of reliability to satisfy system requirements. Fortunately, this is a function which the machine itself is extremely capable of handling, through the means of a "monitor" or "executive" routine.

The monitor program constitutes the highest echelon of control within the machine. It dictates the order of execution of the various programmed functions within the machine, according to their relative priority. It performs a number of bookkeeping tasks, such as assignment of storage in high-speed memory, selection of input/output units, and communicates with the human operator required to change relative priorities, mount tape reels, change paper, cards, etc. Any forced interrupts of machine operation by external system components are handled by this routine, as are the various error correction procedures.

In order for the monitor program to effectively perform these functions, it is necessary that it be immune to almost all of the various catastrophic events caused by any operating program. These are of two types: The offending program may destroy vital data, or other programs. It may fail to complete its execution due to a hang-up in a loop, at a stop, or at an illegal instruction. It is therefore necessary that suitable provisions be made, in machine design, for a "safe" area of storage for at least enough program to regenerate the monitor program, and a means of returning to either the monitor program itself or to a short program which will regenerate the monitor program if it has been lost. This return must be made automatically in case of detectable errors, and in any event, after a given period of time has elapsed.

Detailed specifications for this monitor program cannot be given here. It is, however, fundamental to the integration of the computer into the control system, and its detailed design should be given high priority. With a properly constructed monitor, possessing the two properties mentioned above (it must always be "safe," and it must never lose control of the computer for more than a short time) the data processing subsystem will be able to perform its function of supplying control information under all conditions short of complete machine breakdown.

Major Program Subsystems

Although the monitor program is in control of the machine at all times, it does little if any work itself, but instead calls on the major program subsystem to perform the tasks required. These subsystems themselves have many of the characteristics of a monitor program, but on a much more limited scale. A brief description of the operation and requirements of each of these follows.

A. Operational program.

There is little to be said about the operational program except to enumerate some of its functions and requirements. Normally, this will have the highest priority except during periods of machine malfunction. It requests batches of messages from the Traffic Control Center on a periodic basis, services messages of high priority on a request basis, and prepares displays on both a request and periodic basis. It presents alarm conditions when they are detected. The program must also possess the ability to update itself as the nature of the control system changes, or when errors in programming are detected. Finally, it must be capable of "catching up" if the central processing system has been inoperative because of machine or communication breakdown.

B. Program Checkout.

Included in the program checkout system are the various assembly and compiling programs which the individual programmer may wish to use. The assembly program, in addition to translating between the programming pseudo-language and machine language, performs the functions of detecting as many mistakes as possible before the program is actually run, supplies any additional utility routines desired, supplies any operational data that may be required, runs the program, and presents the results of this run along with debugging information in the original programming language. It should also supply information as to the machine time used.

At least as important as the above, it provides a means for forcing conformity to the rules of the system which have been set up. It fundamentally assists rapid and smooth program changes and improvements by providing a source of information about all programs which have been integrated into the subsystems, and insures system continuity.

C. Planning Programs, etc.

The writers of these programs must also be aware that their programs are to be integrated, on a time-shared basis, with other programs, and must thus observe certain rules and conventions. While they may be relatively independent when first conceived and run, they may eventually be required to operate as part of the operational program, and should be constructed with this thought in mind. Conformity to the system will make any such integration relatively easy.

D. Maintenance Programs.

While the maintenance and diagnostic programs are not normally written by programmers familiar with the control system, the writers of these programs must be aware that the dictates of the control system requires that they should integrate their programs as part of the over-all operating system whenever possible. These programs may be run during periods of low activity, by the monitor program, and should receive information from the monitor as to which components of the machine are available to it, while still allowing for interruption by the monitor program when a high priority request is to be serviced.

E. System Training and Exercising Programs.

The system training and exercising and training subsystem will be composed, in the main, of the same programs which make up the operational program. Although it will be running in real time, and using a large part of the computer system and display system, its operation must not preclude immediate return to the operational status of the control system. Both the operational program and the training programs must be capable of distinguishing between real and simulated data, as well as between operational and exercise requests.

F. Inter-control System Communication.

Although no specific requirements have been made to date, the entire control system may be required to communicate with other such systems at a later time. The possible requirements of such interaction should be studied to insure that this will be relatively easy to accomplish.

Organization of Computer codes into programs and subsystems.

The various classes of programs required for the system as a whole are in themselves large and complex. In order to be able to construct these efficiently, they must be built from a series of submodules, each of which performs a task of a more specific nature. These modules in turn would be constructed from smaller modules, etc. The lowest level of these routines would, of course, be constructed from basic machine instructions in as efficient a manner as possible, with as little duplication as possible. This method of successively breaking the over-all program into subsections corresponds to building a machine out of standard modules and provides a means for easily analyzing the functions performed at each level.

Rigid standards must be set for the construction of these subprograms, and for the means by which they are integrated to perform more complex functions. Such a set of standards can be formalized, and comprise the "Program Design Manual," similar to the "Machine Design Standards Manual" used by machine manufacturers. Such a manual should fulfill the following objectives:

1) Any portion of the program can be understood and changed by any programmer that has access to the manual.

2) Conformity to the standards can be checked by the computer itself whenever possible.

3) Any portion of the code can be changed at any time, with relatively complete assurance that a) parts of the code which should not be affected are indeed undisturbed, and b) all parts of the code which should be updated are indeed updated.

4) The various portions of the code can be manipulated by the monitor program, which will be controlling the over-all execution of the program.

5) All documentation will be supplied and will be in standard form.

6) Programs may be more easily checked out by first checking out the lowest level of routines, combining a set of these to make a higher level routine, combining a set of these, and so on.

The above requirements for the logical construction of a code corresponds to the way in which a machine is built. Resistors, transistors, etc., are first packaged into circuits which are to perform a standard function, e.g., OR gates, AND gates, triggers, etc. These standard packages are then used to build flip-flops, adders, etc., which are then packaged into accumulators, index registers, etc. It should be noted that while any given machine register, e.g., the accumulator, might be designed to do its job faster and cheaper by starting directly with resistors, etc., the dictates of ease of testing, maintenance, reliability, and system documentation far outweighs this approach.

The routines with which the program is built up are characterized by:

1) The process the routine performs
2) The input data required
3) The output data generated.

There is no necessity for a routine which uses this subroutine to know in detail the means by which it performs its task, but it is necessary that a programmer be able to find out this information when required. If the routine calls on more specific routines each of a less complex nature, this is easily accomplished.

In order to be able to construct a program on this master-routine, subroutine, sub-subroutine basis, it is extremely necessary that both the hardware of the machine and the assembly or compiling process permit programs to be constructed on this basis as easily and as efficiently as possible. The mechanism for constructing these routines should include:

1) A mechanism for leaving a sequence of instructions, executing a subsequence, and returning to the main sequence upon completion. The programmer must not have to write more than the bare minimum of instructions to do this. Also, there must be a means of doing this in a recursive fashion (essentially, the routine must be capable of executing itself as a subroutine) for the

following reasons: A subprogram may be interrupted by a high priority request, whose execution requires the use of the same subprogram that was interrupted. Thus, the routine is effectively being executed by two programs at the same time, yet sequence of control must not be lost. Also, experience with extremely complex information processing programs has shown that this recursive ability is extremely desirable.

This requirement may easily be satisfied by the use of a so-called "push-down" memory cell in which to store information about the sequence of program execution. This type of cell has the property of being able to save many words: when a new word is stored in the cell, the words already there are "pushed down." When a word is recalled from such a cell, the word previously stored there is "pulled up" and is once more available. The cell is, in effect, a group of cells with one address and the property of "first in, last out." In use, the location of an order calling for a subroutine is placed in the cell, and control is transferred to the subroutine. To return to the sequence which originally called for the subroutine, the top word in the cell is "pulled up" and control returned accordingly to the higher level routine. The various problems caused by interrupts and the large amount of "bookkeeping" normally required of the programmer in building a hierarchy of subroutines is thus automatically supplied.

It should be noted that while the above functions may be taken care of by machine language coding or by compiler techniques, it is felt that they are so fundamental to the construction of a large, complex, data processing code that if at all possible they should be incorporated into hardware. This appears to be extremely feasible in the MPD computer.

2) In order for the monitor program to be able to manipulate the various routines, there are several desirable properties which these routines should have. First, they should operate independently of their location in memory. Although it is easy to do this in general, with the proposed machine, the assembly program must check to see that this is indeed the case. Further, the assembler should be written, if possible, to supply the proper instruction modifiers necessary so that the programmer need not be so burdened. It should be remarked here that the proper machine instruction modifiers be available (e.g., indirect addressing and indexing, addresses modifiable by the instruction counter, etc.) so that this is always possible if the routine is written in standard format.

Secondly, all references by the routine must be rigidly controlled, and only certain types of these allowed. These are: references to itself, references to the data on which it is operating, references to constant data, references to data of a constant nature particular to this routine only, references to input/output equipment and finally, references to other routines. Each type of reference must be recognizable by at least the assembly program, and perhaps, in some cases, by the operating monitor program. A special class of symbols should be adopted for each of these, and conventions for their use specified. (The use of these special classes considerably aids the programmer not only in understanding other programs in the system but also in writing and debugging his own routines.) References by the routine to itself, including its necessary constants, should be processed by the machine so that the routine will work any place in memory. References to the data of the problem should be made indirect so that the monitor may assign this storage dynamically during the execution of the program. The same applies to input/output device selection, when this is not handled by "system" subroutines. References to other routines must be only through the standard way of executing a subroutine, as described above, and the monitor must make sure that the desired routine is in memory at the proper time.

Finally, the dictates of the recursive property that all routines must have requires that routines never modify themselves and, probably, that some type of push-down storage be available for working storage and index registers. Here again, although this property may be achieved through proper programming, it is such a basic property of all routines that hardware should accomplish this wherever possible.

In addition to the above techniques, many more considerations must be taken into account before the goals of the programming system begin to be realized. These can only be broadly referenced here, but a precise set of definitions should be made before programming is begun on actual operational routines.

Among these is the maximum permissible length of any routine. In a system whereby a one, two or three instruction subroutine is easily implemented, programs may be constructed from a set of basic functions combined with special subroutines, and perhaps a routine need never exceed 30 to 50 orders. Perhaps a standard type of flow chart can be specified for each routine from the basic functions of loop, branch, etc. So called generator routines can be used to apply a given set of functional routines to a set of data, and the programmer need not be concerned with many of details of loop initiation, testing, etc. In such ways, small, basic subprograms can be used to implement many of the bookkeeping details which at present are time consuming and sources of error, and a large part of the programming job will consist of integrating already coded and checked out routines to perform the processing required.

1. The Master-routine subroutine concept should be applied as the basis for extensive modularization of the system.

2. A Monitor routine shall be in control at all times to handle interruptions, control the execution of the different sub-programs, communicate with the operators, etc.

3. Basic function modules shall be specified and coded by experienced programmers.

4. The logical control problem and the required machine functions are much more important than the processing operation codes.

5. Routines shall be capable of recursive execution.

6. Several classes of push-down storage should be available.

7. All control and data specification should be indirect whenever possible.

8. Data and routines should be relocatable dynamically.

9. Routines should never be modified dynamically.

10. Rigid restrictions and rules shall dictate what a coder may write down.

11. Lockout features shall be available to insure that the monitor can keep control.

12. There shall be a programmable clock interrupt feature in hardware.

13. The machine configuration and order list should never be changed unless a great improvement in over-all operating efficiency.

14. No program should assume data block length or data item size to be fixed.

15. Only a very small subset of programmers know all machine functions--the programmer is effectively removed from the machine.

16. A program design handbook shall define rules, procedures, standards, and terms in as few pages as possible.

17. The machine should operate in a non-stop mode at all times except periods of machine malfunction.

18. The complex information processing capability of the system is at least as important as speed.