UNITED ARAB REPUBLIC

THE INSTITUTE OF
NATIONAL PLANNING

التخطيط القومي

5

Memo No. 876

PROCESS CONTROL SERIES

PROCESS CONTROL SOFTWARE

By

ALWALID ELSHAFEI
OPERATIONS RESEARCH GROUP

March 1969

CAIRO
3. MOHAMED MAZHAR - ZAMALEK

INTRODUCTION.

Process environment dictates the design of process computer software. This environment may involve chemical synthesis, metal rolling, steam power production, or any process susceptible to computer control.

Functions of a process computer include such things as reading sensors, setting control devices, and logging operational data while operating as a standard data processor. Design of the software which directs the computer's operation is dictated by the specific requirements of the process involved.

The early expectation that process control computers would require little software support (because they would be programmed once and for all to do a specific job) was very quickly replaced by the realization that not only was all of the sophistication of conventional software required but, in addition, the real-time aspects of the process control application also had to be supported by software. This realization has led to a complete time-shared executive system with an extensive library of subroutines for communication between the computer and process. During process free-time, these executives permit use of the computer for off-line tasks under the control of a subroutine called the off-line monitor to distinguish it from the executive itself. The library of subroutines represents, in a sense, a language for process control, allowing programming of quite extensive real-time control systems. Several software designs requirements are common to all systems, they are:

1. Computer programs must react quickly to process events. Alarms and corrective controls must be given with little delay.
2. Programs must be scheduled so that computing time does not exceed available real time.
3. Input and output devices must be allowed to operate at (or near) their maximum speeds.

4. Actions which occur at definite times or within definite time periods must be scheduled according to a real-time clock.
5. Information must be gathered from the process as it becomes available and transmitted to the process as it is needed.

Process control software may best be understood by considering the control problem itself and the software and hardware needs it generates.
A process computer system in a chemical plant gathers temperatures, pressures, and flow rates from the process. A raw material composition and marketing requirements change, programs calculate and send out changes in plant operating conditions. When emergencies occur - such as extreme temperatures or pressures - visual and audible alarms and corrective actions are sent to the plant. Process information, recorded each hour, helps determine long range operating strategy.

In a metal rolling process, information is gathered beforehand from laboratory analyses, physical measurements, and customer requirements. As a rectangular slab of metal is converted to a flexible strip, its position and temperature are recorded through sensors. Data are combined by the program, and corrections in roll force, speed, and temperature are calculated and sent to the mill. Alarms are generated when schedules or design requirements cannot be met or when equipment fails.

When starting up or shutting down a steam power plant, the computer system gathers turbine speed, motor position, temperatures, and pressures. As turbine speed is increased, the program scans eccentricity and vibration. If vibration exceeds an upper limit, turbine speed is held constant or decreased, or the turbine may be shut off. Visual alarms are sent to the operators to inform them of the emergency condition. Once conditions are steady, the system monitors temperatures and vibration and prints them each hour.

CONTROL LEVELS

Since the objective of the control systems is to successfully operate the process despite the presence of many disturbances, it is appropriate to partition the disturbances according to their relative frequency and consider the control of the system in the presence of each of these disturbances separately. The resulting control system hierarchical in form, is shown in Fig. 1

The highest frequency disturbances which must be considered are physical upsets which cause process variables to deviate from their "reference" values. For example, flows, temperatures, pressures, and the like will not long stay constant without the continual intervention of a control of some kind. Such control is usually fast and, if accomplished by a digital computer, termed direct digital control (DDC), and is generally regulatory in nature. Often this first level of control is accomplished by analog controllers with the digital computer supplying only the higher levels of control (such a control system is termed a supervisory control system). This is especially true in existing plants which already have analog control and add the supervisory control in order to increase production, cut costs, etc. The operator communication task in Fig. 1 is concerned mainly with this first level of control. One of the most important tasks at this level is alarming in case any process variable exceeds prescribed safe limits. This involves informing the process operator and then either taking appropriate action or turning control over to the operator. The computer is especially efficient in alarming and displayingoverloads and is a tremendous assist to the human operators.

The second set of disturbances of importance are less frequent in nature and are caused by the changing mode of operation of the process. For example, a power system has drastically different load requirements during day and night hours and the operation of the process differs

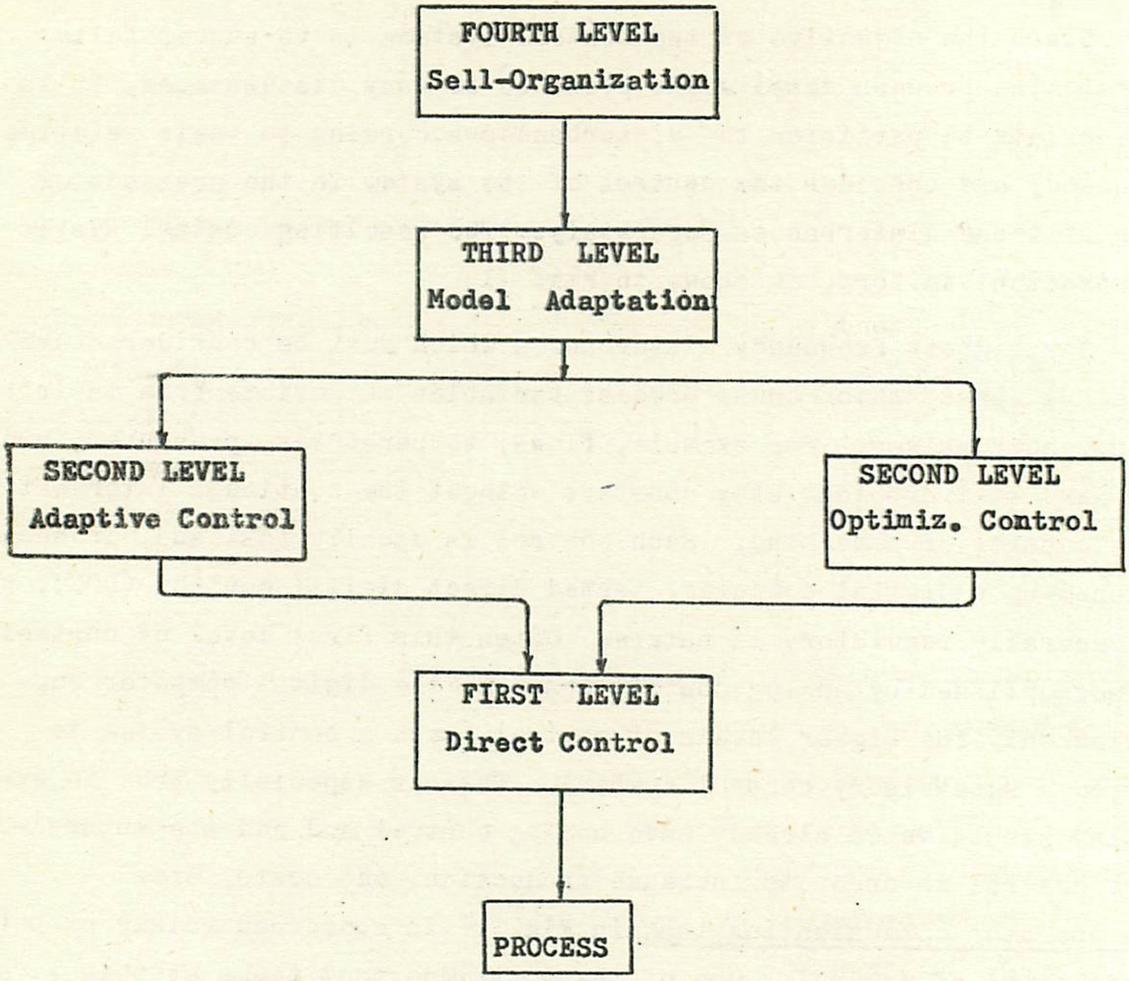


FIG.1 DIFFERENT CONTROL LEVELS

considerably at high and low loads. The second level control system which compensates for these disturbances has two functions. First, as the disturbance changes, it may be desirable to change the operating points (reference values for process flows, pressures, etc.) in order to make the operation as economically attractive as possible. Such a control system supplies new reference points to the first level analog controllers or digital controllers and is termed an optimizing controller. Secondly, on the same level, the dynamics of the process itself may change as the operating conditions of the process change, requiring modification of the first level controllers if adequate dynamic performance is to be retained. Adaptive control is implemented on this level to update or adapt the parameters of the first level direct digital controllers. This is most easily done if the first level of control is DDC rather than analog control since the controller parameters are then merely numbers stored in machine memory.

The third class of disturbance is even slower in frequency and consists of changes in the parameters of the process itself. This may be due to seasonal changes, aging, corrosion, and many other factors.

If the optimizing control is to be effective, it must have an accurate mathematical model and consequently the objective of the third level controller is to make use of process data to update or adapt the parameters of the mathematical model of the process itself.

The highest level of control compensates for the lowest frequency disturbance of all, changing process structure (in contrast to changing process parameters). This fourth level of control is seldom automated but rather supplies appropriate information to management and operators of the process so that they can make the proper decision as to structural changes in the overall control system and process (Fig. 1); This level might include emergency restart of the process, emergency shut down, etc.

Observe that this breakdown of the problem results in hierarchical control system in which each level of control effects directly the level of control directly below it and in turn is controlled by the level immediately above it. This structure is attractive not only from a conceptual viewpoint, but also because each level can be designed somewhat independantly of each other level, thereby permitting the design of a complex control system by means of building blocks which are easily changed and improved upon as the appropriate technology becomes available.

From a software point of view, this division of a large task into many independent subtasks is very desirable since the programming of each subtask may be done independently and is then easily documented and updated at later times. However, the various subtasks are not necessarily performed at the same rate or even in the same sequence. In fact, various tasks may be performed under emergency conditions or upon demand of management or the process operator. Consequently, the software must permit the programmer to efficiently control the sequencing of these tasks and to change the sequence easily when it is desired. Of particular interest is the observation that the computational load is approximately the same for each level of control. That is, going up in level decreases the frequency at which the computation must be performed, but the complexity of computation increases with the result that the load (product of computation time and frequency) remains about constant.

The design requirements of the controlled process systems dictate the following program design characteristics:

- Since both time and memory are limited, programs share both.
- Some programs voluntarily give up time and space to other programs when it is necessary to wait for input or other actions.

- Some programs involuntarily give time to other programs which operate input/output devices, allowing the devices to operate at or near rated speeds.
- A real-time clock is used to schedule actions which must occur at definite times or time intervals, and to generate a time-of-day display. There must be a provision for resetting the clock through an external device.
- Core memory is fast but expensive, and bulk memory (drum or disc) is slow but relatively cheap. Programs in the system arrange for sharing the limited core memory between the programs in bulk sotrage.

PROGRAMMING FOR DIFFERENT CONTROL LEVELS:

There is a significant difference in the programming requirements of the first and higher levels of control. The simple algorithms of first level control are applied to processes in which there are many variables, all of which are regulated in a similar fashion. Consequently the computational load is very repetitive in nature and efficient programming and core allocation are necessary. On the other hand, the higher levels of control are relatively complex and nonrepetitive in nature and it is more advantageous here to use an algorithmic language in order to provide flexibility and good documentation.

The extensive input-output facilities of a process control computer may be needed by any one or all of the levels of control, since all make use of process data during their operation. Consequently,many decisions must be left to the programmer to make for each particular installation. For example, rate of scan of analog inputs, type of scan (sequential or random), changes in scan rate, strategy in case of input or output error detection, etc.., all vary from application to application and in fact from one level of control to another witin a given application. Thus any software system necessarily must permit the programmer to communicate easily with all of the hardware in the computer and cannot incorporate arbitrary decisions about these problems in a single executive.

Most of the software systems currently available for process control are designed to support the higher levels of control (the so-called supervisory control levels) rather than the first level, and are in the form of an executive system with real-time Fortran as the basic language. Two factors permit such executives to be used with supervisory control systems. First, the speed of machines has significantly increased so that Fortran-level programs can compete in terms of speed with machine-language programs in the earlier machines. Secondly, secondary storage has become readily available, permitting large executive systems and efficient storage and saving of programs outside of core. The problem of servicing the first level of control can be solved without sacrificing the executive if a two-computer system is used: one computer doing primarily first level control (DDC) and the second, under control of the executive system, performing superivsory tasks as well as backup of the DDC computer.

CONTRAN LANGUAGE:

One exception to the real-time Fortran approach to process control languages is the CONTRAN system being developed by Honeywell. This language is an outgrowth of the Consequent procedure language developed by Fitzwater and Schweppe. Their language called TASK 64, is a modification of ALGOL 60 to include task processing statements and consequent procedures (procedures which are initiated when prescribed conditions are fulfilled). Such a language operates within an executive system as does Fortran, but goes a step further than the subroutine library approach. Control of sequence of various portions of the control program is obtained by the specification of set of Boolean variables or switches for each program so that the program will be executed when, and only if, these conditions are fulfilled. Thus these conditions, rather than the order of program statements or routines, determine the sequence of their operation. Control of interrupts and communicationbetween the process and the computer through input-output devices are obtained through subroutine calls as in real-time Fortran.

CONTROL PROGRAMS :

Program sequence control (PSC) - controls the sequencing and initiates the loading and execution of user-specified process core loads.

Master Interrupt Control (MIC) - automatically determines the type of each interrupt as it is recognized and transfers control to the proper interrupt servicing routine.

Interval Timer Control (ITC) - provides a programmed real-time clock, a timer for TSC, nine programmed interval timers, and control for two machine-interval timers.

Time-Sharing Control (TSC) - Controls the timesharing of variable core between process and nonprocess core loads.

Error Alert Control (EAC) - provides the following functions whenever an error occurs:

1) Optionally saves core for future reference,
2) Optionally branches to a user's program for further error analysis.
3) Prints an error message.
4) executes a specified recovery procedure.

Communications Control (COMC) - controls communication with the PSC and the I/O programs.

Bulk Transfer Driver (BTD)

controls transfer between bulk storage and core

SEQUENCE CONTROL:

Statements which permit the programmer to control the order in which tasks are performed interrupts serviced, and off-line jobs permitted. Such control is important, since the various levels of control are necessarily carried out in sequence rather than in parallel and the order is critical. For example, a sequence of tasks might be to collect certain data, use a statistical identification routine to determine parameters of the process, and finally to use an adaptive routine to change the controller parameters. An optimizing routine too large for core can be executer in parts if the programmer has control over the sequence of programs.

Program Sequence Control (PSC) is a control program that handles the flow of control from the mainline core load to the next. PSC functions are initiated by execution of PSC CALL statements in the user's program. The specific functions of PSC are:

1- Execute the next sequential mainline core load. The new core load overlays the one that contained the call.

2- Save the mainline core load in progress (on disk) and load a special core load for execution.

3- Restoresthe core load that was saved in item 2 and continue execution from where it left off(the statement following the CALL SPECL).

4- Queue mainline core loads associated with interrupts whose occurrence has been recorded.

5- Execute the highest priority mainline core load listed in the core load queue.

6- Insert mainline core load entries into or delete them from the core load queue.

For PSC to perform the above functions, a CALL statement must be executed for each one. The specific CALL statements and their parameters are described below.

Commands giving this type of control can be categorized in three groups used to:

1- CALL the next mainline core load to be executed.

2- SAVE the present mainline core load (or disk) and CALL a special mainline core load for execution.

3- RESERVE and CONTINUE execution of the saved mainline core load consequently, those commands can be classified as follows:

1- CALL STATEMENTS, including:

- Normal Call - CALL CHAIN (NAME), specifying next program to be executed.
- Special Call - CALL SPECL (NAME)
- Return Saved Mainline - CALL BACK

2- QUEUING STATEMENTS, including:

- Insert Into Queue - CALL QUEUE, entering program in a waiting queue.
- Delete From Queue - CALL UNQUEUE, removing program from a waiting queue.
- Execute Highest Priority Core Load - CALL VIAQ
- Queue Core Load If Indicator Is ON - CALL QIFON
- Clear Recorded Interrupts - CALL CLEAR

3- SHARING STATEMENTS, including:

- SHARE, indicating availability of free time in which non-process programs may be executed under the control of the off-line monitor.

Such statements may be freely inbedded within process programs written in FORTRAN. Through use of these commands within programs, the programmer can control the frequency and order in which the various levels of control are performed. Even when various levels are not performed on a regular basis (for example, when a certain function is performed only upon operator demand), these commands permit control over the sequence. In response to an operator-initiated interrupt, the interrupt subroutine can decode the request and call for the appropriate program to be entered in the queue and then executed when it has the highest priority. Of equal importance is the ease by which sequence is changed as the process control problem changes with time.

INTERRUPT CONTROL:

This includes the control of the routines which service interrupts and the control of the interrupts themselves. For example, during certain routines it may be advantageous to delay the serviving of interrupts to minimize exchanges of programs or to prevent certain interrupts entirely (as when a routine cannot be used recursively and may be called from more than one level.

The program in charge of such control is the MASTER INTERRUPT CONTROL program (MIC). It controls the servicing of interrupts, an interrupt may occur at any time but it will not be recognized by MIC unless the interrupt is on a level that is not marked and is of higher priority than the present level of machine operation. The user-assigned interrupts can be delayed from being recognized by masking the level to which they are assigned. The servicing of process and COUNT subroutiions can also be delayed by recording their occurrence. There are, basically, two types of interrupts:

EXTERNAL INTERRUPTS:

are those associated with the process and programmed interrupt features. They are serviced, or recorded, by one of four types of user - written routines:

1- Skeleton Interrupt Routine
2- Mainline Interrupt Routine
3- Interrupt Core Load
4- Mainline Core Load

INTERNAL INTERRUPTS:

are those associated with:

1- I/O Devices Control

2- Interval Timer Control, for four types of timers:

   a. A timer for time-sharing control

   b. A programmed real time clock

   c. Nine programmed interval timers

   d. Two machine intervals timers

3- Trace Control

4- Error Control, some of whose functions are:

   I Optionally, dump core storage to disk

   II a. If in a nonprocess program: Terminate the program if the error cannot be operator corrected.

   b. If in a process program: branch to the user-written error subroutine that is with the core load.

   III Update error counters maintained on disk

   IV Execute a subroutine for the device or error condtion print an error message on the printers, and set up possible recovery action.

Commands to permit this control include SPECL (to stop the program in progress, save it in secondary storage, and execute another program), BACK (to service interrupts which were not serviced at the time they occurred) and CLEAR (to ignore interrupts which occurred but were recorded rather than serviced immediately). These commands are most useful within interrupt subroutines which can determine the present status of the process (alarms, overloads, etc.) and decide what actions to take, including complete restart of the programs, aborting of certain actions, or even turning control over to the operator. Control over the interrupts themselves implies actual inhibiting or allowing of the interrupts to occur. This control is obtained through commands such as MASK, UNMASK, SAVE-MASK, and RESTOREMASK, which inhibit or allow specified levels of interrupts and permit determination of the status of the interrupt levels (inhibited or not) at any time. Through selective use of masking, data channels can keep operating for the transmission of data in and out of core while process interrupts are inhibited until the short routine in progress is complete (assuming the hardware permits this). These masking control commands must be available to the programmer, for only he decides in a given situation whether or not it is permissible to delay servicing of certain interrupts from the process. These commands give the programmer the opportunity of increased efficiency of execution of his program, but also place on him the burden of insuring that essential process functions not prevented by indiscriminate masking of interrupts.

INPUT/OUTPUT CONTROL:

This division includes statements which create communication between the process itself and the computer. These statements are included in what is known as I/O SUBROUTINES (which are included in the SUBROUTINE LIBRARY). These subroutines were originally designed to reduce the amount of time spent by the programmer in accomplishing the input and output of data from and to the various input/output devices attached to the computer. They handle all of the details peculiar to each device (including complex interrupt functions) and are capable of controlling many I/O[*] devices at the same time. The subroutines permit the programmer's attention to be directed to the problem-solving aspects of each individual job rather than regulating different I/O units.

[*] I/O devices = Input/Output devices.

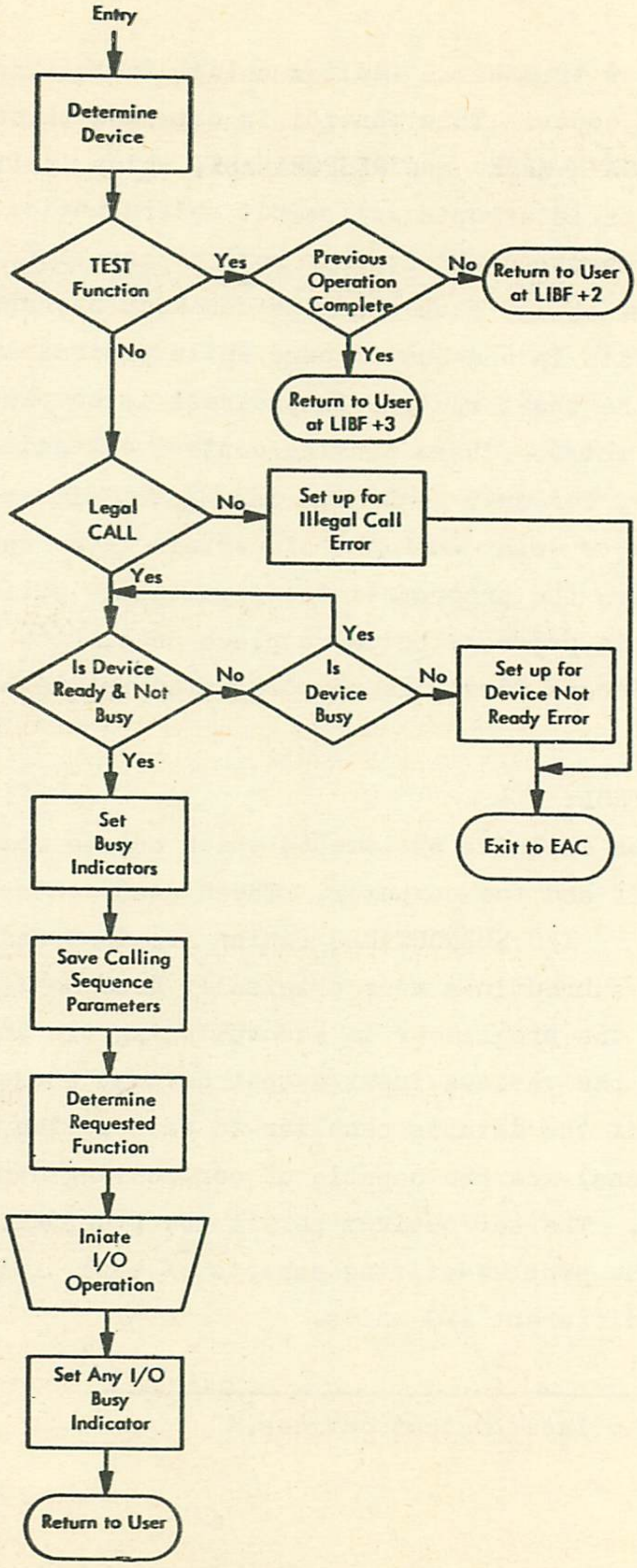

Entry
Determine Device
TEST Function
Yes
Previous Operation Complete
No
Return to User at LIBF +2
Yes
Return to User at LIBF +3
No
Legal CALL
No
Set up for Illegal Call Error
Yes
Is Device Ready & Not Busy
No
Is Device Busy
Yes
No
Set up for Device Not Ready Error
Exit to EAC
Yes
Set Busy Indicators
Save Calling Sequence Parameters
Determine Requested Function
Iniate I/O Operation
Set Any I/O Busy Indicator
Return to User

FIG2: CALL ROUTINE

The following characteristics are common to all the I/O subroutines, they are:

METHODS OF DATA TRANSFER, they include:

- Direct Program Control
- Data Channel

I/O SUBROUTINE OPERATION, describes briefly the internal makeup of the I/O subroutines, they include:

- Call Routine, with the following functions:

  1- Determines if any previous operations on the specified device are still in process.

  2- Checks the calling sequence for legality

  3- Saves the calling sequence

  4- Initiates the requested I/O operation

  Fig. 2 is a typical illustration of such a routing

- Interrupt Response Routine, which is entered as a result of an I/O interrupt, it functions as follows:

  1- checks for errors

  2- does any data manipulation

  3- initiates character operations (and, retry operations in case of errors)

  Then, returns control to MIC which then returns control to the user.

GENERAL ERROR HANDLING PROCEDURES :

These routines categorize the error and choose an error procedure. Errors belong to either of two categories

- Errors detected before an I/O operation is initiated.
- Errors detected after an I/O operation has been initiated Accordingly, there are two types of checks to be performed:

- Pre-operation checks
- Post operation checks

BASIC CALLING SEQUENCE:

Each of the subroutines is entered via a calling sequence. These calling sequences follow a basic pattern. There are some parameters which are common to most of the subroutines, such as:

- Name parameter
- Control parameter
- Special condition parameter
- I/O area parameter

ANALOG TO DIGITAL CONVERSION ROUTINES:

At any level of control, data from the process may be neccesary, requiring that any program be able to request analog or digital input data from specified locations in the process. Commands which permit this are various subroutine names which are programmed to convert one or more analog signals into digital form and store them in prescribed locations or to take prescribed variables, convert them to digital form and output them to the process. Such routines make use of the INTERRUPT STRUCTURE of the computer and DATA CHANNELS (if available) in order to perform such tasks asynchronously.
These subroutines include:

- CARD SUBROUTINE
- DISK SUBROUTINE
- PRINTER SUBROUTINE
- PRINTER-KEYBOARD SUBROUTINE
- PAPER TAPE SUBROUTINE
- PLOTTER SUBROUTINE
- DIGITAL INPUT SUBROUTINE
-
-

- DIGITAL/ANALOG OUTPUT SUBROUTINE
- ANALOG INPUT-SEQUENTIAL SUBROUTINE
- ANALOG INPUT-SINGLE READ SUBROUTINE
- ANALOG INPUT-RANDOM READ SUBROUTINE

Extensive presentation for each of these subroutines is behind the scope of this article, still an example is given with respect to the IBM-1800 COMPUTER(*).

(*) For full details refer to the "IBM 1800 Time-Sharing Executive System.

DIGITAL/ANALOG OUTPUT SUBROUTINE

This subroutine transfers digital/analog information to a number of addresses, under direct program control or data channel control. Table-chaining is permitted on the data channel; however, continuous scanning is not permitted.

Control Parameter

This parameter consists of four hexadecimal digits used as shown below:

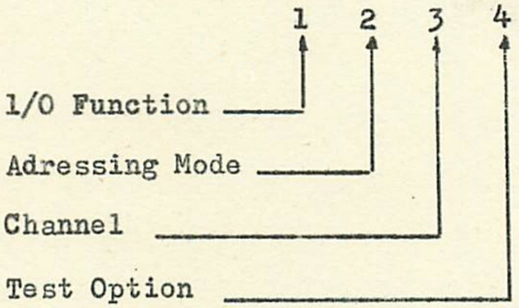


FIG 3

I/O Function

The I/O function digit specifies the operation to be performed by the digital/analog output subroutine. The functions, associated digital values, and required parameters are listed and described below.

Function	Digital Value	Required Parameters(*)
Test	0	Control
Write	1	Control, I/O Area, special condition
Write Pulse	2	Control, I/O Area, special condition
Write Buffered	3	Control, I/O Area, special condition
Reset	4	Control

(*) Any parameter not required for a particular function must be omitted.

Test. Depending on the test option, branches to LIBF+2 if test indicates busy, or to LIBF+3 if test indicates not busy.

Write. Writes the requested number of digital/analog values. If direct program control is specified, no interrupts are involved in the operation. Therefore, the subroutine does not return control to the user until the entire operation is complete. After each value is written, when using direct program control, the word count is checked. If the requested number of values has not been written, the subroutine writes the next value. If the requested number has been written, the subroutine returns control to the user. This routine only operates in the sequential mode.

Write Buffered. Same as write, except that the control for buffered output is given after the write function has been performed.

Reset. Resets all digital/analog output operations in progress and resets all indicators.

Addressing Mode:

The addressing mode digit specifies one of four addressing options for data channel operations only.

0 - Random addressing
1 - Single addressing
2 - Random addressing with external synchronization
3 - Single addressing with external synchronization

Channel:

This digit specifies the method of data transfer used for this operation.

0 - Direct program control
1 - Data channel control

Test Option:

If zero (0), DAOP tests to see if the previous call has been completed. If one (1), it tests to see if the pulse output timer is on.

I/O Area Parameter:

The I/O area parameter is the label of the control word that precedes the user's I/O area. If the function is direct program control, the word count is the number of points to be written, plus one for the first DAO address. If the function is data channel control, the mode determines what the word count should be. If the mode specifies random addressing, the table contains interleaved digital/analog addresses and data to be written. Each address precedes its data word. If the mode specifies single addressing, the table contains one address followed by all the data words to be written for that address.

The word count is equal to the number of address words and data words in the table. If the mode indicates random addressing, the word count is twice the number of data words to be written, since there is an address for each data word. If the mode indicates single addressing, the word count is the number of data words to be written plus one, since there is only one address word. The subroutine expects the 16-bit digital value for output to be in the following format.

Bits	0	Sign
	1- 13	Data bits (DAC Models 3 and 4)
	1- 10	Data bits (DAC Models 1 and 2)

Special Condition Parameter:

If the I/O area and the I/O subroutine are in the system skeleton, the special condition routine must be in the skeleton.

MEANS OF ERROR DETECTION:

Because all of these subroutines make extensive use of data channels, device indicators, and interrupts, the form of the subroutines themselves is not under the control of the programmer. For example, a call for an analog input may find the multiplexor busy, in which case the subroutine may simply wait until it is free or return to the calling program ...... depending on the particular routine. Detection of an error in transfer of data may result in repeated attempts to read the data, the number of attempts being built into the subroutine. In some application these subroutines might not be suitable because of the speed at which they run, the arbitrary way they respond to error conditions, etc.

It is, of course, possible to provide additional simple subroutines which test indicators for various error condition and to call these before attempting to use a device. An extensive set of such routines is available in THE GENERAL ELECTRIC GE/PAC EXECUTIVE SYSTEM, including routines for saving machine indicators and registers, An important device is the interval timer, a counter which may be set up to cause a program interrupt at any desired time. This permits a watchdog type of operation for maximum reliability. For example after performing some operation, an interrupt after a given length of time may be requested which will call out a program which determines if the operation actually was carried out.

The interval timer can be used to sequence certain operations by requesting an interrupt after a specified time interval followed by execution of the task. In the hier-archical control system, the higher the level, the less often the tasks are performed. Frequency of operation of the optimizing function, for example, may depend upon the detection of disturbances by a lower level and/or may be done at fixed time interval through use of the interval timers.

COMMUNICATIONS.

Associated with the executive software is a family of subroutines and interrupt drivers which allows the programmer to communicate using standard calling sequences.

These subroutines provide for:

(1) Initiation of one program by another

(2) Timing, either by interval or time of-day.

(3) Linkage between interrupts and programs.

(4) Requests for input or output of data.

(5) Status checks on input/output devices and request-progress.

(6) Store and fetch from bulk storage.

Communication between a specific application program and the executive software is effected through standard calling sequences clearly documented for the user. For programmers working at the compiler level, the same calling sequences are generated either by the equivalent compiler input/output statement or by some extension to the compiler language.

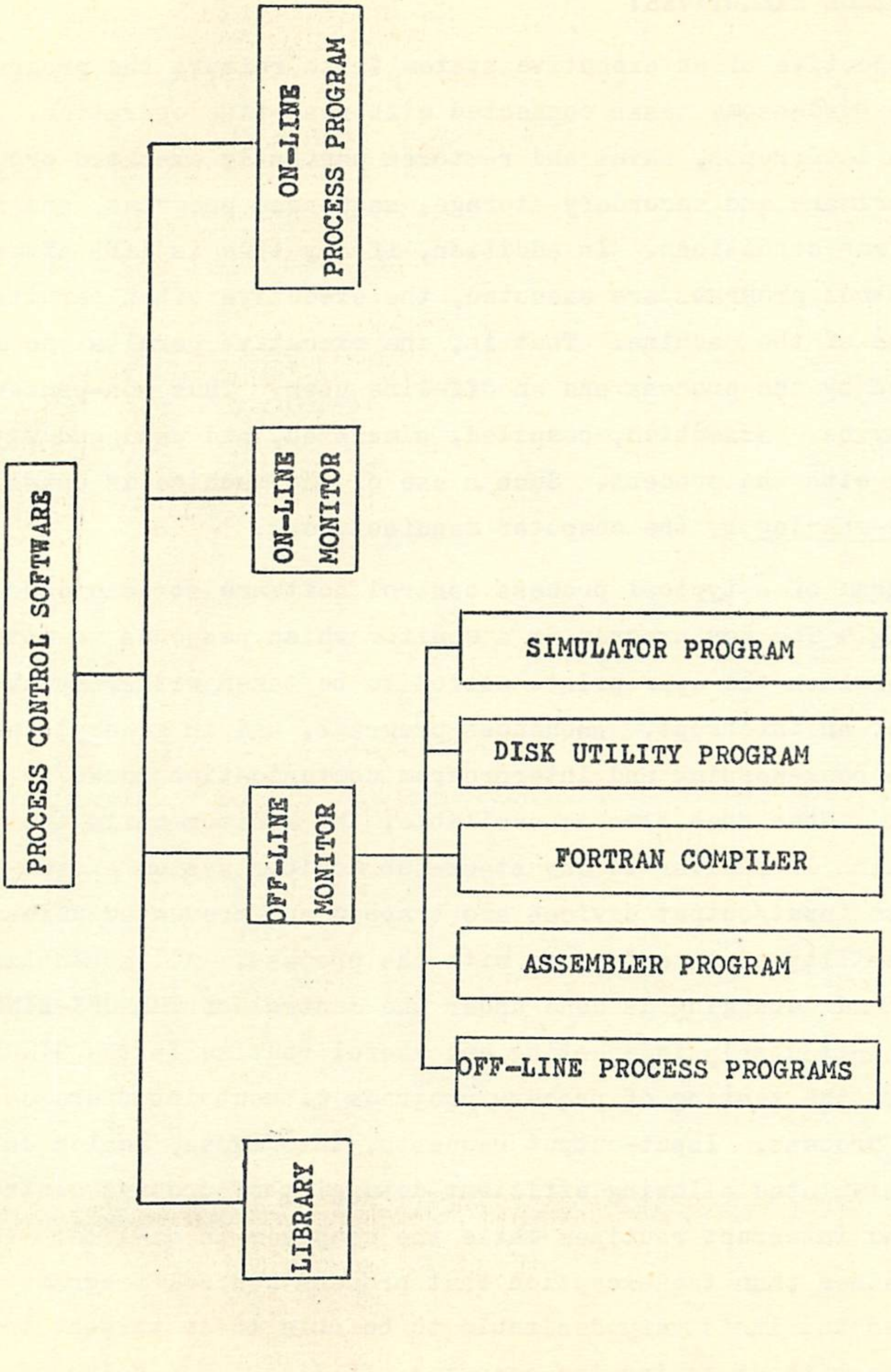

PROCESS CONTROL SOFTWARE
ON-LINE PROCESS PROGRAM
ON-LINE MONITOR
OFF-LINE MONITOR
LIBRARY
SIMULATOR PROGRAM
DISK UTILITY PROGRAM
FORTRAN COMPILER
ASSEMBLER PROGRAM
OFF-LINE PROCESS PROGRAMS

FIG 4

PROCESS CONTROL EXECUTIVES:

The objective of an executive system is to relieve the programmer of the more burdensome tasks connected with real-time operation. It responds to interrupts, saves and restores partially executed programs, allocates primary and secondary storage, sequences programs, and responds to error conditions. In addition, if any time is left after the process control programs are executed, the executive often permits off-line use of the machine. That is, the executive permits the machine to be shared by the process and an off-line user. Thus non-process programs may be assembled, compiled, simulated, and debugged without interfering with the process. Such a use of the machine is usually called <u>time-sharing</u> by the computer manufacturers.

A diagram of a typical process control software structure is shown in Fig.4 The key program is a monitor which responds to interrupts, determines the appropriate action to be taken with respect to servicing of an interrupt, sequences programs, and in general takes care of the book-keeping and interprogram communication tasks in a real-time system. When free time is available, the monitor calls the off-line monitor, which is similar to any stack-job monitor system except that all calls to input/output devices are trapped and prevented unless there is no possibility of interference with the process. All assembling, compiling, and debugging is done under the control of THE OFF-LINE MONITOR.A particularly interesting and useful routine is the SIMULATOR, Which allows the testing of process programs without interference with the actual process. Input-output requests, interrupts, analog data, etc. are simulated allowing efficient debugging of process control programs and interrupt routines while the computer is on-line. It is the rule rather than the exception that process control programs change often and it is very desirable to be able to do this at the installation without taking the computer off-line. The value of changes made at the installation far outweights, the slow running speed of such simulators. Within the off-line monitor are the usual utility routines which permit the control of programs stored in secondary memory and the changing of programs which are used for on and off-line process control.

The main task of theexecutive is the running of user written process control programs which necessarily are greatly concerned with the many input-output devices. Although the executive cannot permit direct programming of input-output devices, it does permit their control through calls to a library of real-time subroutines. The combination of the fortran language and these library routines results in something akin to a process control language which permits programming of any of the higher control problems.

Elements of this language vary, of course, among different computers, but the basic functions are similar.

NONPROCESS MONITOR:

The nonprocess monitor provides the user with a programming tool that simplifies the task of generating, organizing, and testing programs executed under control of the 1800 TSK System and to supervise execution of nonprocess programs. The nonprocess monitor can be operated on-line under control of the system director, or it can be operated off-line. Off-line operations are possible after system generation. Initial assembly of certain system programs and user's beginning routines are done under TASK control.

The primary function of the nonprocess monitor is to provide continuous processor-controller operation during a sequence of jobs that might otherwise involve several independent programming systems. The monitor coordinates the processor controller activity by establishing a common communications area in core storage, which is used by the various programs that make up the monitor. It also guides the transfer of control between monitor programs and the user's programs. Operation is continuous and setup time is reduced to a minimum, thereby affecting a substantial time saving in processor-controller operation and allowing greater programming flexibility.

Control records, which are used to direct the sequence of jobs without operator intervention, must be prepared prior to the actual operation. The control records are included in a stacked input arrangement.

The nonprocess monitor (Fig. 4) is composed of five programs:

(1) Simulator Program. This program is designed to provide a means for simulating a process control or data acquisition program without interfering with the normal operation of the process. Each input/output call sequence of the program being simulated is analyzed and simulated individually.

Various options are offered throughout the simulator. For example, an analog input call sequence can obtain input from either cards or a random number generator. The option chosen is specified in a control record read by the simulator program.

(2) Disk Utility Program (DUP). DUP is a set of routines designed to aid the user in performing the functions of disk maintenance. That is, it is capable of storing, deleting, and outputting users programs and defining system and machine parameters. DUP updates the location equivalence table (LET) when a change is necessary and also maintains other communication areas.

(3) Fortran Compiler. The compiler translates programs written in the FORTRAN language into machine language and automatically provides for calling the necessary arithmetic, functional, conversion, and input/output subroutines.

(4) Assembler Program. The assembler translated programs written in symbolic language into machine language. Basically, it is a one-for-one type assembly program. That is, the assembler usually produces one machine language instruction for each symbolic instruction of the source program. Provision is also included for the user to easily made use of input/output, conversion, and arithmetic subroutines that aré a part of the subroutine library.

(5) Off-Line Process Programs. This is a group of programs whom are executed during process-free time. They include programs for different managerial and accounting purposes such as:Management Decision Making Techniques, Statistical Analysis of Obtained Results, Different Balance of Company Activities, Files Updating (Record-Keeping System, ...etc.)

The operation of those five programs is perfectly organized using a master program called the "SUPERVISOR PROGRAM". It decodes the monitor control records in the stacked input for nonprocess Jobs and calls the proper monitor program to perform the desired operation. For example, a typical sequence of jobs might be execution of a payroll program,compilation of a FORTRAN process control program, and simulation of a core load that includes the program just compiled. The supervisor program calls the program loader, FORTRAN compiler, and simulator program, respectively, to handle these jobs.

JUSTIFICATION OF EXECUTIVE USAGE:

The supervisory executive described above is a very significant advance in process control software and should service the needs of a majority of control installations. What are the disadvantages of such executive systems which prevent their application at times ? Three types of installations are not efficiently serviced by these executives.

The first are the very small systems which cannot justify the additional core storage and secondary storage necessary to use the executives. Many installations (at least in the past) started out small and, consequently, with special programming systems written especially for that job. Later they grew large enough to justify the use of an executive system. However, it is difficult to decide to scrap a great deal of working software in order to use an executive. Even though in the long run it might be the best solution. Often a middle of the road solution which is not satisfactory to anyone is chosen.

The second type of installation which cannot directly use the executive is a system which is primarily first-level (DDC). Efficiency, reliability, speed, and quick response time are the objectives here rather than simplified programming and convenience. As mentioned, when this function can be delegated to a separate computer with its own soft-ware the executive can still be used in the supervisor computer. This may be the solution in the long run as indicated by several installations in which this is now being applied.

The third exception application is a critical process where special programming requirements are necessary. For example, the nuclear reactor control system supplied by Control Data placed so great an emphasis on reliability that no executive now available could efficiently satisfy the requirements. Of importance here is that such unique applications will occur more and more often as the volume of process control applications grows and some software support must become available.

MEMORY			
FIXED AREA		EXCHANGABLE AREA	
ON-LINE USAGE	INTERUPT ROUTINES APPROPRIATE TO TASK (Others In Secondary Storage)	SKELETON EXECUTIVE (Including Some Input/Output Routines)	SOME INTERRUPT ROUTINES

FIG 5: Memory Arrangement For Process Control Computers

With sufficiently large core storage (8K minimum) one of the executive systems can respond efficiently to process demands. A possible allocation of core is shown in Fig. 5 The skeleton executive program is permanently resident in core and includes all routines necessary for communication, with secondary storage and the servicing of interrupts with very short response time requirements (mostly high speed data transfer devices). The non-permanent portion of core storage includes process programs and interrupt routines whose response time is slower than that of the permanent routines. Any interrupt which is not part of the skeleton executive system or loaded with the process program is serviced by exchanging the process program for a service routine in secondary memory followed by restoration of the interrupted program. To minimize secondary storage exchanges, such routines are usually not interruptible except by interrupts whose service routines are in the skeleton executive. Thus there are basically three response times for interrupts and the programmer may choose the combination which is best for a given application.

REAL-TIME IMPLICATION:

Implementation of such a control system in real time requires certain hardware which significantly affects software requirements. Most important of these are the analog input and output hardware which requires relay and, possibly, solid state multiplexing devices together with a multilevel priority interrupt system. Since reliable operation of the control system is very important, it is necessary to make maximum use of the hardware to insure that the control system will seldom fail. For example, it is inexcusable to permit failure because of a card reader jamming or a relay multiplexer receiving an illegal address. Hardware interrupts and indicators are provided to check such contingencies and to obviate them. This of course complicates tremendously the software requirements since they generally must be written for the "worst case" situation. Moreover the slow operating speed of the analog multiplexor (50 to 100 points per second, for instance) and the need for data in real time necessitates careful consideration of the real time control programming problem.

As control systems (such as those described here) were implemented, it became apparent that not only was the control system not to be programmed once and then run for many years but rather that it would probably be changing constantly throughout the life of the process and control systems. This is due in part to increasing knowledge of the process which permits better control systems to be designed for various levels, and also to the ever-changing characteristics of the process itself as it undergoes improvements, new instrumentation is added, and the like.

Early real-time process computer systems used drum computers with no automatic interrupt ability. The programs produced included a rudimentary form of time sharing in which peripheral devices were actuated at intervals. The intervals were determined by counting drum revolutions at coding time.

The first drum/core process computers, which included automatic priority interrupts, introduced new dimensions to process programming. Time sharing of core memory between programs is standard. Systems of executive, timing and input-output control programs were developed and refined until a reasonably standard set called a real-time monitor or control executive was produced. Currently, time and space sharing are well established in the process computer field. Recent advances in other areas should contribute greatly to future development of process computer systems.

SPECIAL CONFIGURATIONS OF PROCESS CONTROL COMPUTERS:

Basic components of a computer system are: memory, arithmetic unit, input, and output.

Process computers differ from free-standing scientific computers in memory arrangement and input/output. They resemble scientific computers in most other ways: types of memory, instruction repertoire, speed of execution, and input and output devices.

Process computers, almost always binary machines, usually have word lengths of 12 to 24 binary bits and most arithmetic is performed in single precision (one word per number). In contrast, many scientific computers have word lengths of 32 bits or more, allowing great accuracy to be maintained. However, data received from process sensors usually preclude the need for extreme accuracy. Most process computers use magnetic drum or disc memories - not only as backup and file storage, but also as a dynamic extension of the faster, more readily accessible cores. While some recent scientific systems are using this approach for time sharing, the more traditional scientific machines use bulk storage only at the beginning and end of program runs.

In addition to such conventional input and output devices as card readers and punches, type writers, line printers, and cathode ray tube displays, process computer systems include equipment which communicates directly with the process and its operators. The primary input devices for process computers are analog and digital scanners. The analog scanner a matrix of relays coupled with an analog-to-digital conveter and other signal conditioning equipment - operates under program control to transfer digitized process measurements into the computer memory. Such measurements are used as process control feedbacks as well as for data logging. The digital scanner allows contact information such as open or closed as well as panel settings and the contents of digital counters to be transferred directly to computer memory.

Output from the computer to the process is in the form of analog signals (voltage) or the opening or closing of contacts. This information, sent through an output controllere allows the computer system to maintain control over the process. Digital output is also presented to operators in the form of warming lights, horns and numeric displays. Concerning memory configurations,we need sufficiently large core storage (8 K minimum) so that the executive can respond efficiently to process demand. For memory arrangement, a possible allocation of core is shown in fig. 5. The skeleton executive program is permanently resident in core and includes all routines necessary for communication, with secondary storage and the servicing of interrupts with very short response time requirements (mostly high speed data transfer devices). The non-permanent portion of core storage includes process programs and interrupt routines whose response time is slower than that of the permanent routines. Any interrupt which is not part of the skeleton executive system or loaded with the process program is serviced by exchanging the process program for a service routine in secondary memory followed by restoration of the interrupted program. To minimize secondary storage exchanges, such routines are usually not interruptible except by interrupts whose service routines are in the skeleton executive. Thus there are basically three response times for interrupts and the programmer may choose the combination which is best for a given application.

OTHER CONTROL SYSTEMS:

Beside the previously described control system, there are several other control systems being used in the field of process control.

Two of these systems are the following:

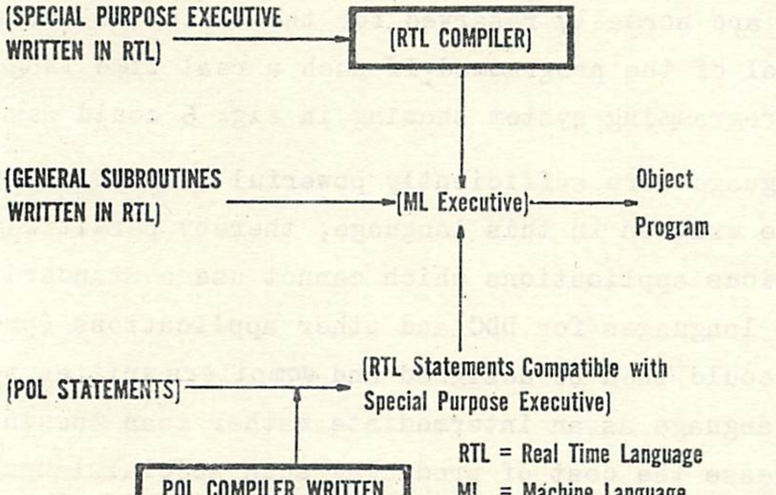
1- Fixed Core Areas . A process computer system in which programs have fixed core areas may be similar in many respects to the system described above. Timing, input, output, and program selection are not signigicantly different. However, core memory allocation is much more restrictive since each program must be assigned a permanent starting location. The starting location is not assigned at execution time, but at assembly time. If one program is being executed in a given core area, other programs which must operate in this same core area are effectively locked out. At times of high computing load, this may impose a severe restriction on system response time. Fixed core areas are used with computers which do not have the ability to automatically execute programs regardless of their memory locations.

Both the fixed and floating core area programming systems have several programs resident in core at any time, except the one being executed in a partial state of completion. These systems tend to reduce the numbers of bulk/core transfers per unit time.

2- Single Program in Progress. A programming system which requires a much simpler executive program for memory and time sharing allows only one program to be in progress at a time. The program runs from initiation to completion with interruption only for input/output actions. Response times will normally be slower than in the systems described above. The only noticeable advantages of such a technique are potential simplicity and the low memory requirement of the executive program. These systems may be entirely adequate where rapid response to process demands is not needed.

Process control programs provide the framework for a process computer programming system. Such programs operate in a real-time environment. They respond to process, operator and time demands. Their access to computer memory and input/output devices is limited by other system demands.

A systematic approach to process programming provides a way to meet the requirements of a computer controlled process. Other benefits of this approach are: maximum use of the available core space; ability to write programs with minimum attention to the real-time environment; communication links for all programs whithin the system, and be transferred from one system to another.


(SPECIAL PURPOSE EXECUTIVE WRITTEN IN RTL)
(RTL COMPILER)
(GENERAL SUBROUTINES WRITTEN IN RTL)
(ML Executive)
Object Program
(POL STATEMENTS)
(RTL Statements Compatible with Special Purpose Executive)
POL COMPILER WRITTEN IN RTL
RTL = Real Time Language
ML = Machine Language
POL = Process Oriented Language

FUTURE TRENDS:

With the rapid increase in the number of computer control installations and the shortage of qualified systems engineers and programmers to install these systems, process control software will quickly develop in the direction of further simplification of the programming problem.

A primary and essential step towards this might be the development of a true process control language. The main difficulty with the Fortran library system of subroutines is that the programmer has no control over the inner workings of the executive system and the interrupt and error handling characteristics of the individual subroutines. It might be desirable to develop a real time process control language which is algorithmic in nature and sufficiently powerful to produce the executive itself together with the subroutine library. Such a real - time language must permit control over the allocation and freeing of core and secondary storage specification of interrupts and responses to interrupts and other functions which are normally reserved for the executive rather than being under the control of the programmed;If such a real time language were availabe, the programming system showing in fig. 6 could result.

If the language were sufficiently powerful and efficient the executive could be written in this language, thereby permitting easy changes for various applications which cannot use a standard executive. Special purpose languages for DDC and other applications (process oriented languages) could then be designed and compilers written to produce the real-time language as an intermediate rather than machine language. This would decrease the cost of producing such special languages and decrease the problem of obsolescence because of hardware changes.

The importance of simple flexible software in DDC cannot be overemphasized,, since it must be used by phant enginneers, for example, in all process phases from startup through regular poant operation. A number of manufacturers are considering a DDC compiler system whose input language is a block diagram specification of the control algorithms to be implemented. The output of the compiler would then be a machine language program which could run by itself in a DDC computer. Such a block diagram language is similar to digital simulation languages such as Midas and pactolus. Changes in the first level of control would then merely require perhaps a recompilation of the block diagram statements. One advantage of this approach: this type of programming is familiar to plant engineers, since it is used in analog simulation. This approach is very attractive in that different sets of blocks could be supplied for different industries and applications, making it very easy for customers to make use of the computer. The problem is one of economics. Although the language may not differ much from one application to another, the implementation may be quite different because of diferent operating speeds required for scanning and output of data. Consequently, a different compiler might be required fordifferent applications, making the approach uneconomical.

Within whatever DDC software is supplied, there may be included a routine which permits communication with a supervisory computer operating under an executive system. Such systems are now availabe. For example, the PCP-88 system supplied by Foxboro is made up of two PDP-8 computers with Foxboro interfaces and a complete real-time Fortran executive system for the supervisory computer and DDC software for the first level computer. Two big advantages of such an approach are the separation of the programming problem into two parts, with each servied by the most convenient language ... and the increased reliability of the system because of the duplication of central processors.