INTRODUCTION

A major advance in the design of computers was the provision of buffered or autonomous peripheral transfers which enabled computation to proceed concurrently with a transfer. In many computations, however, the total saving obtained is quite small; for instance, 10 milliseconds of computing overlapped with 100 milliseconds of transfer provides a gain of only 10% and the central processor must remain idle for the remaining 90 milliseconds. An extra burden is also placed on the programmer who must arrange his work so that computation can take place while the transfer is occurring. This is enough of a task when the program is written in machine language but, when the program has to be compiled, the compiler has to be considerably more sophisticated in order to reach the same efficiency and, as a result, compilation will take longer. The problem boils down to an economic one of weighing the cost of program production against the cost of machine time gained by the extra efficiency.

One step away from this dilemma is to use magnetic tape as the sole input and output medium for the computer and to provide off-line equipment for the transcription of data to or from magnetic tape. The transfer rate of magnetic tape is much higher than that of punched cards or printers and the loss of time during transfers is thus reduced. This method, however, requires extra off-line equipment which may be expensive and lack flexibility.

An alternative approach is to enable the central processor to store several programs simultaneously, and, each time a program is forced to wait for a peripheral device, to enter an alternative program which is at that time able to proceed. In this way the efficiency of the overall installation is increased since the central processor is kept working for a higher proportion of the time and a larger set of available peripherals can be kept operating. This method does not require any special off-line equipment and is very flexible in use.

In order to obtain system efficiency, one method to adopt might be to arrange, at the time of programming, for a number of jobs to run concurrently. The problems encountered with this method are connected with implementation and are the same as those found with a single program system using autonomous transfers, but increased in complexity by several orders of magnitude. Once the marriage between the several jobs has been consummated, it is practically indissoluble and the same group of jobs always has to be run at the same time. This may be perfectly satisfactory in certain special applications but generally this method is found to be too restrictive.

Alternatively, programs can be provided with suitable latching arrangements, so that,
although written separately, they may be joined together to make an efficient entity. However, the programmer still has to be very conscious that his program will be associated with a number of others when being run and has to imbue his program with a social instinct, which is tedious and not always easy to do.

Therefore, in spite of some loss of central computer time in order to perform organizational functions, it seems advisable to be able to write programs with no thought that they may be sharing the computer with other programs and to have a supervisory routine to make all the switches between the programs. This system of operation we refer to as "time-sharing."

As an example of time-sharing on a computer, consider the example of three programs which are, in order of priority:

Program P1. A routine which reads data from punched cards and stores it on magnetic tape, the contents of two cards being stored in a single tape block.

Program P2. A routine performing a magnetic tape sort with a simple keyword.

Program P3. A routine performing a long calculation using no peripheral equipment at this stage.

The basic rule of operation is that the program with the highest priority is entered whenever possible. A program which is waiting for a peripheral transfer to be completed is said to be "suspended." In Diagram I a time chart is shown which demonstrates how these three programs share the time of the central processor one with another. In order to demonstrate as many program interactions as possible, the time scale has been deliberately distorted.

In a recent paper, it has been stated that a minimum of five requirements must be provided in order to have an operational time-sharing system, unless it is to be restricted to jobs which are specifically designed to be run concurrently or to fully debugged programs (but, except for trivial cases, it is impossible to know when programs are fully debugged since they are similar to scientific theories—they can only be disproved). These requirements are:

1. Memory protection to prevent one program from destroying others.
2. Program and data relocatability so that the same routine can be used in different locations at different times.
3. A supervisory program.
4. An interrupt system.
5. Symbolic addressing of peripheral devices.

In this paper we hope to show how these requirements have been met in the FP6000 system by means of a combination of hardware and software.

THE FP6000 COMPUTER SYSTEM

The FP6000 is a fast medium sized computer system constructed in a packaged, modular form. At the centre of the system is a general purpose digital computer with a minimum core store of 4096 words which may be increased in modules of 4096 words to a maximum of 32,768.
words. The machine operates in the binary mode and a word, of length 24 bits, can be used to represent an instruction, a signed integer or fraction, or four 6-bit alphanumeric characters. The store cycle time can be either 2 or 6 microseconds. The computer is a parallel machine with a clock rate of one megacycle.

A large variety of peripheral devices can be added to the computer, making the system versatile and adaptable to commercial data processing, scientific and real time applications. The peripheral equipment is broadly divided into two classes: character at a time, for example paper tape readers, and word at a time, for instance magnetic tape. For each type of equipment in either one of these two classes, the interface between the peripheral device and the computer is made the same. Due to this modular method of attachment, it is easy to expand any system to meet increasing demands and wider applications.

A program called EXECUTIVE organizes all peripheral transfers and the allocation of the central processor's time to the various programs being run simultaneously. This program is written in such a way that the programmer does not have to consider time-sharing or the mechanism of peripheral transfers when writing programs. In addition, EXECUTIVE provides certain macro-instructions for carrying out floating-point arithmetic and enabling the programmer to have a master program which time-shares with one or more sub-programs.

The system is controlled from a console which has deliberately been made simple. Basically, it consists of an electric typewriter which enables the operator to give instructions to EXECUTIVE and EXECUTIVE to pass messages to the operator. A large number of these messages are concerned with manual action which is required on the peripheral equipment. In addition to the typewriter, the console has six buttons: to call the attention of EXECUTIVE to an incoming message on the typewriter, to cancel such a message and so on. A paper tape reader and punch, when incorporated in a system, are also located on the console.

THE CENTRAL PROCESSOR

The core store of the central processor is not divided into blocks in any way; however, each program in the machine at any one time has a definite area of store allocated to it. A region of core store is allocated to a program by EXECUTIVE; this is done by specifying two addresses, the starting address known as the "datum" and the final address called the "limit." The core store area between a program's datum and limit is called the program's reservation. Datum and limit points occur only at multiples of 64 words. These addresses are set by EXECUTIVE and stored in special registers in the arithmetic unit during the time a program is being obeyed. In operations where addresses of locations are used, datum is added automatically by hardware to the addresses; because of this, programs are written with addresses relative to the datum and, therefore, are locatable anywhere in the core store. Indeed, during the running of a program, EXECUTIVE may transfer the program to another area of core store in order to make room for a new program. All that needs to be done after the transfer from one area of core store to another is for EXECUTIVE to change the values of datum and limit corresponding to the program. This operation of relocation is only carried out after EXECUTIVE has ensured that no peripheral transfers are in progress.

The first eight core store locations belonging to each program are known as "accumulators" and can be used for arithmetic and counting. Accumulators 1, 2 and 3 may also be used for indexing.

Programs are stored with one instruction per word; for a normal instruction the word is divided as shown:

\[ X \quad F \quad M \quad N \]

where \( X \) is an accumulator address
\( F \) is a function code
\( M \) is an index register address
\( N \) is a core store address or a number.

Obeying normal instructions consists of performing the function \( F \) between the contents of the core store location \( N \) and the contents of the accumulator \( X \) and placing the result in either \( N \) or \( X \) depending upon \( F \).
There are 16 groups of 8 function codes, though not all function codes are allocated. Five of these groups consist of macro-instruction codes: these operations are performed by EXECUTIVE. While a function is being obeyed datum is automatically added to the appropriate addresses in the arithmetic unit and checks are made that these addresses do not lie outside the program's reservation. These checks are performed by special hardware in the arithmetic unit referred to as the "reservation checker." Since a program cannot be allotted less than 64 words there is no necessity to check that the accumulators are within the program's core store area. However, the N-address is so checked.

The operation of a simple order takes place in five phases or beats:

Beat 1.
During this beat the address of the current instruction is sent to the core store and the instruction is obtained and placed in the arithmetic unit. The contents of this core store location are regenerated so that no change is made to the store by this operation. However, if at the end of the beat, it is found that the address of the current instruction did not lie in the area allotted to the program, further accesses to the core store are inhibited, though the remainder of the instruction proceeds as usual.

Beat 2.
If the N-address of the instruction requires indexing, the value of the index register is taken from core store, regenerated, added to the address and the result stored in the arithmetic unit. This beat is omitted where indexing is not required.

Beat 3.
The first of the two operands required for the function is taken from the core store and put into the arithmetic unit. It is arranged that the first operand to be taken from the core store is the one which is unchanged by the function; that is, after being read out it is regenerated in the core store. At the end of this beat, the result of the reservation check on the indexed N-address is examined and, as at the end of beat 1, if the check has failed, further accesses to the core store are prevented.

Beat 4.
The second of the two operands is obtained and, since its value is to be changed, it is not regenerated. The function is performed during this beat.

Beat 5.
The result of the operation is written back into store and, except in the case of branch instructions, the address of the current instruction is increased by one. This address is retained in the arithmetic unit ready for the next instruction.

Then follows the beat 1 of the next instruction. If the reservation check has failed at either of the two points where it was tested a forced entry to EXECUTIVE is made at this time.

There are two modes of operation of the central processor: the Normal mode and the Executive mode. The Executive mode, as its name implies, is used while EXECUTIVE is being obeyed. While in Executive mode, under the control of EXECUTIVE, the current value of the datum may be added to any or none of the addresses N, X and M in an instruction. This enables EXECUTIVE to have access to information within a program's area, using the program's index registers when required. Reservation checking is not performed during the operation of EXECUTIVE and some of the functions in the group of macro-instructions have special actions, though most are unassigned.

The above example of an entry to EXECUTIVE, due to a reservation check failure, might be termed an "involuntary" entry or "interruption." The other reasons for an involuntary entry are:

(a) Depression of a console push button.
(b) An illegal order; that is, one with an unassigned function code.
(c) A monitor point. In order to assist a programmer in developing a program, EXECUTIVE can arrange for an interruption to occur after certain specified types of instruction, for example a successful branch instruction, in a particular program.
(d) The time register has reached zero. This register, which is an optional extra, enables time-accounting to be performed in a service-center machine or interruptions to occur at specific intervals so that an operation can be performed on a special peripheral device.
(e) A peripheral incident, which can be an indication that either a peripheral transfer has been completed or one of the automatic checks on the transfer has failed.

When an involuntary entry to EXECUTIVE occurs, the current instruction of the program being interrupted is dumped in location 8 of that program. The state of the overflow register and certain other indicators particular to the program being left are also stored in the same location at this time, so that, when a return is made to the program, the state of these may be reset correctly. A reason for entry register is set so that EXECUTIVE can find out the reason for the interruption. The machine is then set to Executive mode and the instruction number in the instruction number register is set to the entry point in EXECUTIVE corresponding to an interruption.

When a program is input to the computer a priority number is assigned to it. This number is allocated by the programmer and is specified on the program tape with a number of other parameters such as the amount of core store space, drum space and other peripheral equipment required by the program. The priority number is originally assigned on the basis of the number of peripherals used by the program and the proportion of time spent in computation relative to that awaiting peripheral transfers.

A maximum of four programs may be run simultaneously; however, this is an arbitrary limit set by EXECUTIVE and is not dictated by hardware considerations. When EXECUTIVE was being written, in order to assign store space for certain lists, such a limit had to be established, and it was felt that four was a reasonable number for a machine of the size of FP6000. Since this limit is set by EXECUTIVE, there is nothing to prevent, where the installation warrants it, a special version of EXECUTIVE being assembled with a higher limit. Each program may be divided into a master and at most two sub-programs, which are each assigned an individual priority. EXECUTIVE keeps a record of each program and sub-program, in descending priority sequence. This list is called the Priority List and has at most twelve entries. Also recorded in the Priority List is the present state of each program: free to be obeyed (called Active), suspended awaiting a peripheral, suspended awaiting an operator message, and so on. Whenever EXECUTIVE has completed its required actions, it scans the Priority List for the active program with the highest priority and transfers control to this program. Each program is given a code name which is made up of four alphanumeric characters and is used in all communications between the operator and EXECUTIVE by means of the console typewriter.

During input of a program, EXECUTIVE allots the core store to be assigned to the program according to the core store requirements given on the program tape and the core store available in the computer at the time. In order to keep as large as possible an area of contiguous core store locations free, whenever a program is finished and is no longer required in the computer (the term used is “abolished”), all current programs stored above the program to be abolished are moved down the core store to fill up the now vacant space. When the core store area has been assigned to a program, the appropriate values of datum and limit are stored among the records which EXECUTIVE keeps for each program currently in the computer.

When referring to peripheral units in a program, the programmer numbers his units of a particular type starting from zero. Suppose, for example, that the program named “BILL” uses four magnetic tape units. The programmer will number these units 0, 1, 2 and 3. If BILL is to be run on an FP6000 system with six tape units which are numbered 9 to 14 and if, at the time when BILL is input, units 9 and 12 of the system are already being used by current programs in the system, then EXECUTIVE will allot units 10, 11, 13 and 14 to BILL. EXECUTIVE will type out a message to the operator giving the correspondence between BILL’s unit numbers and the system unit numbers. Every time BILL refers to his magnetic tape unit 0, system unit 10 will be used. Because of this arrangement for the numbering of peripheral units, all peripheral transfers are handled by EXECUTIVE. This method also shields the programmer from a lot of the work in organizing peripheral transfers. If BILL is run on another occasion, the allocation of actual units to the units 0 to 3 in BILL may be quite different, but, since the change from the program's
unit numbers to the system unit number is performed automatically by EXECUTIVE, no alterations have to be made to BILL.

ATTACHMENT OF PERIPHERAL EQUIPMENT

In general, each peripheral device has its own control unit; exceptions to this rule are magnetic tape and drum units, several of which may be connected to one control unit. Each control unit has the address of a special-register assigned to it; this special-register performs the function of a mail box for the passing of control information to and from the control unit. In addition, each control unit has a core store word associated with it which is used by the peripheral control unit to store the transfer address and the count of words or characters to be transferred. The initial value of this control word must be set prior to a peripheral transfer being started. The operation may then be initiated by sending a starting signal to the special-register for the device. The function for sending instructions to the special-registers is available only to EXECUTIVE.

Once a transfer has been started it proceeds autonomously, each word or character being transferred to or from the core store as requested by the peripheral control unit. Between such transfers the central computer is free to obey programs. When a request for the transfer of a word or a character occurs it competes with similar requests from other peripheral units and, eventually, the transfer takes place. There is nothing new in this approach; several of the larger computers use this method. However, the usual system is to have a separate piece of logic in each peripheral control unit to perform the addressing and counting operations required, the central processor thus being able to continue with its normal operations, unhindered except when the peripheral control unit requires access to the core store at the same time as the central computer. When this occurs, a small hesitation in the rhythm of the central computer takes place. Since FP6000 is a medium-sized computer it was decided that, in order to reduce the cost, it would be better to use the central arithmetic unit to perform the addressing and counting functions. The actual amount of time spent by the arithmetic unit in performing hesitations remains a small portion of the total time required for the transfer; for example, with a 2 microsecond core store, during a magnetic tape transfer, the arithmetic unit will be available for program calculation for at least 90% of the time.

When a peripheral control wishes to make an access to the core store it sends a hesitation request signal to the computer control unit. If more than one of these signals occurs at once, a priority system ensures that the faster peripheral units get precedence over the slower ones. The chosen peripheral control is sent a hesitation select signal and, at the end of the instruction currently being obeyed, the next instruction is not started but a hesitation is performed. A hesitation requires 4 beats:

Beat 1.
During this beat the core store address of the peripheral control word is supplied to the core store and the control word is read into the arithmetic unit. A counting operation is performed and, if this is the last hesitation in the transfer, a stop signal is sent to the peripheral control unit. The address part of the control word is augmented to give the core store location to which data is to be transferred during the next hesitation.

Beat 2.
The new value of the control word is written back into the core store.

Beat 3.
The address from the control word is supplied to the core store and the word is read.

Beat 4.
Depending on the direction of the transfer, either the new data word is written into the store or the original data word is regenerated.

Then follows the next order or another hesitation. Magnetic tape units attached to FP6000 have a character transfer rate of 65 Kc/s—that is, one character every 15 microseconds or, since magnetic tape transfers occur one word at a time, one word every 60 microseconds. The net interval is a fraction of this if several magnetic tape units are performing transfers simultaneously. Some of the instructions in FP6000 take longer than 60 microseconds to be obeyed; provision has therefore been made for hesitations to be carried out, if required, at certain defined break-points within the sequence
of basic operations which make up an instruction, as well as between instructions.

The only instructions which have to be broken into to make a hesitation are the longer ones which nearly always involve some sort of loop in the micro-program (for instance multiplication or division), and therefore a breakpoint is arranged in the loop. During the hesitation process only two of the registers in the arithmetic unit are used. If a hesitation has to be made, the contents of these two registers are stored temporarily in core store locations 9 and 10, which are set aside for this purpose. Diagram II presents a simplified flow diagram of the operation. From this it will be seen that only hesitation requests from peripheral controls which cannot wait, fast hesitations, are allowed to interrupt the course of an instruction. When the hesitation operation is completed, and if there are no more fast hesitation requests, the two registers are reset from core store and the long instruction is carried on from the point at which it was suspended.

Diagram II. Simplified hesitation flow diagram.

During the course of a hesitation no reference is made to the output of the reservation checker; to do so would involve making suitable settings in arithmetic unit registers, which would be too costly in time. EXECUTIVE therefore checks, before initiating a transfer, that the transfer does not involve any core store locations outside the area of the program calling for it. This test is very simple to make and involves taking the starting address for the transfer, adding to it datum and the number of words to be transferred, and checking that the result is less than the value of the limit.

If, during a hesitation, a stop signal is sent to the peripheral control unit because the end of the transfer has been reached, the peripheral control unit will terminate the transfer and, when this has been done, send a signal to the central computer. This signal causes an entry to be made to EXECUTIVE at the end of the instruction currently being obeyed. This type of entry may also be classed as an involuntary entry. When such an entry is detected by the central control unit a mark is made in the reason for entry register. This consists of a special register which may be read by an instruction available to EXECUTIVE only. Corresponding to each of the reasons for entry—depression of a console push button, illegal order or reservation check failure, monitor point or the time register reaching zero—there is one bit in this register. The peripheral incident type of entry is expanded so that there is a bit which corresponds to each peripheral control unit. Before entry is made to EXECUTIVE the current value of the instruction number register is stored in core store location 8 of the program being obeyed at the moment of the interruption. The machine is then switched to Executive mode and the address of the entry to EXECUTIVE is forced into the instruction number register.

EXECUTIVE

Diagram III gives a flow diagram of the operations which occur during an involuntary entry. We will call this "EXECUTIVE Entry I."

The first action on entry is to read the reason for entry register; if this is not zero the cause for interrupt with the highest priority is isolated. The bits giving the reasons for interrupt
fall into two main classes: those involved with special action, for instance an illegal function, and those involved with peripheral transfers.

If the interrupt is one where special action is required the necessary action is performed. In the case of monitor action the program development routine is entered to print out the required monitoring information for the programmer. In the case of an illegal operation or a reservation violation the offending program is marked in the priority list as being suspended. When a program is suspended for either of these reasons it is marked in EXECUTIVE's records as awaiting a message from the operator. The operator is informed, by means of the console typewriter, that the program has been suspended and the reason for this suspension is also given.

If a peripheral incident is the reason for interrupt, the actual control unit may be identified from the reason for entry register. With this information, the special-register associated with the peripheral control unit may be interrogated. Since the reasons for an interrupt can vary with the type of peripheral, a separate subroutine is used for each type. For example, a paper tape reader can cause an interrupt, not only at the end of a transfer, but also if the parity of the character read from the paper tape is incorrect or if there is no paper tape to be read. In addition to the three bits used for signalling these three conditions there is also a bit used to signal the fact that the reader is busy. The subroutine which deals with the type of peripheral causing the interrupt examines these signals by reading the special-register associated with the peripheral control unit, thereby clearing the indicator bit in the reason for entry register, and tests if the transfer has been successful. If so, any program which has been suspended because it is waiting for this peripheral is reactivated. If the transfer has not been successful, the program making the transfer is suspended and a typewriter message is output.

In the case of the interrupt being due to the end of a message from the console typewriter, EXECUTIVE has to take action immediately on this message. The types of message which can be input on the typewriter are:

1. **GO #BILL**

   This message will reivate program BILL and allow it to continue from the instruction number contained in its core store location 8, where the instruction number at the time of interrupt was stored. There are two main uses for this instruction: to start a program which has just been read in, since it is normal to hold a program in suspension immediately after input to allow its data to be set up on various peripheral devices; and to continue a program which has been suspended awaiting some operator action, such as changing a tape reel.

2. **GO #BILL AT 1234**

   This message will reactivate program BILL and allow it to continue from instruction number 1234. This allows a program to have optional entry points and may also be used to facilitate restart procedures following a peripheral failure.
3. LOAD #BILL ON X
This informs EXECUTIVE that a new program is awaiting input on peripheral unit X. Provided that there are not four current programs already in the machine and that at least 64 words of core store are free, EXECUTIVE will read the first part of the new program to determine whether the requirements for core store and peripherals can be met. In either case a message will be output and, if the new program can be accommodated, EXECUTIVE will read it in under a binary read routine. This input operation will be time-shared with the other programs. The peripheral units of the installation each have an absolute number by which EXECUTIVE identifies them; these numbers are prominently displayed on the units.

4. LOAD #BILL ON X, Y
The LOAD message may optionally request Y words of core store, in which case this request overrides the store request on the program tape. This version of the LOAD message is most useful for programs such as matrix schemes for which the store required varies considerably with the specific data to be used.

5. SUSPEND #BILL
This suspends program BILL awaiting a further typewriter message. This message is mainly used in an emergency, for example, if the program seems to have got out of control and is continuously reading the same section of magnetic tape.

6. DELETE #BILL
This abolishes program BILL from the store and lists. Normally a program will be terminated by a special instruction in the program and the typewritten message is only used in special cases such as the program having gone out of control or if the computer is required for an urgent job for which there is insufficient space.

7. ALTER #BILL AT 1234/FXMN
This changes instruction number 1234 of program BILL to FXMN. This provides a means of altering a single instruction during program development.

8. CONTINUE #BILL ON X
This enables a further part of a program to be read in or the input of a program to be continued after a peripheral failure.

9. MONITOR #BILL ON X
This message informs EXECUTIVE that a program development tape is to be read from unit X.

10. PRIORITY PRINT
This is a request to EXECUTIVE to print out the current contents of the priority list.

11. REVISE PRIORITY #BILL YY
This provides the facility of changing the priority rating of a program so that an urgent job can be accelerated, at the cost of a reduction in efficiency for the overall system.

When the input typewriter message or the peripheral incident has been dealt with, a test is made to determine whether there are any further reasons for interrupt. This is done by checking the copy of the reason for entry special-register retained by EXECUTIVE. If there are any further reasons the process is repeated. If there is none the Time-Sharer is entered. This scans the list of programs and enters the active program with the highest priority. If there is no program which is active, a loop back is made to read the reason for entry special-register. This loop will continue until a peripheral incident occurs to make a program active.

There is a second entry to EXECUTIVE: this is from a program entry by means of the macro-instructions. There are four types of these:

1. Peripheral transfers.
2. Organizational instructions.
3. Master and sub-program instructions.
4. Floating-point operations.

Diagram IV is a flow diagram of EXECUTIVE for this entry. When an instruction which falls into one of these categories is encountered in a program the control unit of the computer first stores the current instruction number in the program's core store location 8, then transfers the indexed value of the N-address to core store location 1 in EXECUTIVE.
core store area and transfers the instruction itself, not indexed in any way, to core store location 2. The machine is then set to Executive mode and the entry address to EXECUTIVE for a voluntary entry is forced into the instruction number register.

EXECUTIVE first examines the function which caused entry. Since not all the functions which cause entry to EXECUTIVE have been allocated, provision is made for an illegal instruction. In this case the program is suspended and a message to this effect is printed out on the typewriter.

For peripheral transfers, more information is required by EXECUTIVE than can be accommodated in one word. The actual transfer instruction is only one word, but the N-address (which may be indexed) specifies the address in core store of the control word, formed by the program, for the transfer. This word contains a counter of the number of words to be transferred and the core store address for the start of the transfer. The type of peripheral involved is defined by the function digits of the transfer instruction, and the particular program's unit number is given in the X-address. Before the peripheral transfer instruction is obeyed, though not necessarily immediately before, a mode of transfer must have been set by a special instruction provided for this purpose. This mode controls whether, for example, a reading or writing operation is to occur on magnetic tape. EXECUTIVE checks that the transfer lies within the program's reservation and that the program has, in fact, a unit X. If either of these checks fails the program is suspended and the operator informed. The unit is then checked to see if it is already busy, in which case the program is suspended waiting for the peripheral to finish its current transfer. If the transfer can be made, EXECUTIVE takes the information provided in the instruction, forms the appropriate control word and initiates the transfer. The program is then re-entered.

There are a number of instructions available concerned with the organization of peripheral transfers. As we saw just now, the initiation of a peripheral transfer does not cause automatic suspension of a program and it is up to the programmer to ensure that he does not cause any conflicts by neglecting this fact. The philosophy of this method is that it allows the programmer to arrange to be working with one set of data while the next set is being read or the previous set output and, at the same time, removes the need for costly hardware to check that a programmer is not using data before it has been read. An instruction of the form "Suspend this program if its unit X of type N is busy" is provided to enable the programmer to avoid conflicts.

Another instruction in the class of organizational instructions is "Suspend this program pending an operator message to EXECUTIVE." An instruction of this type will usually be preceded by the output of a message on the console typewriter, telling the operator of an action that is required on a peripheral unit (for instance, changing a reel of tape). The output of this message is under the control of the program. The program may, in this message, mention its own unit number or may ask EXECUTIVE what absolute unit number has been assigned on this occasion.
Apart from the facility of being able to perform time-sharing between independent programs, FP6000 is able, by virtue of EXECUTIVE, to have a program time-sharing with different parts of itself. We have already seen how, by use of the instruction to EXECUTIVE "Suspend this program if its unit X of type N is busy" a programmer is able to avoid overwriting information that is involved in a peripheral transfer. However, this may lead to the central computer being held up waiting for a transfer when in fact it could be performing useful work. Consider a program which is producing results which are to be printed on a line printer and suppose that, because of the nature of the computation, the results are produced in groups of ten lines, none of which can be printed until all are computed. This situation can easily arise in the production of certain types of table. If we adopt the straightforward approach, the sequence of events will occur as follows:

1. Compute group of results.
2. Print group of results.
3. Return to step 1.

Even if we assume that there are other programs in the machine which can use the arithmetic unit while the results are being printed, the printer is still idle for part of the cycle, so that it is not being used efficiently. This situation can be avoided by splitting the program into a master program and sub-program which time-share with each other. If we divide the core store originally allocated to the program into an area containing the master program, an area containing the sub-program and an area of working store, and also arrange that the datum and limit of the master program are such as to include the whole of the program's core store area and that those for the sub-program just include its own area and the working store area, the master program can be made to time-share with the subprogram. Because of the time-sharing ability of the computer, no computation can be timed absolutely, relative to a peripheral transfer. It is therefore necessary for the programmer to take suitable precautions so that it is not possible for overtaking to occur. The master program must not be allowed to overwrite results with new ones before the previous set has been printed, nor must the subprogram output a set of results before they have been completely processed. Similar situations arise with input sub-programs.

Three instructions are provided to enable the programmer to organize the time-sharing between his program and sub-programs without difficulty. When the master program reaches a point at which it is ready to initiate the sub-program, for example when it has some results which are ready to be printed, an instruction of the form "Activate my sub-program X entering at instruction N" is obeyed. The value of X can be 1 or 2, allowing a master program to have two sub-programs. This instruction causes EXECUTIVE to activate the sub-program, set the sub-program's instruction number equal to N and enter the time-sharing routine. Note that the sub-program is not necessarily entered at this time; it is simply activated within the priority list. Usually, because the sub-program is using peripheral equipment, the priority of the sub-program will be higher than that of the master. The master program remains active so that it time-shares with its own sub-program, and EXECUTIVE will pass control initially to the part with higher priority.

When the master program has a new set of results to be printed, but wishes to check that the output sub-program is ready for them, an instruction of the form "Suspend me if my sub-program X is active" is obeyed. If the sub-program is still actively engaged in printing the last set of results, the master program will now be suspended to enable the sub-program to catch up. If, however, the sub-program has completed its task at this stage, it will have made itself inactive by the instruction "Suspend me awaiting reactivation by the master program, and reactivate the master program if it is waiting for me." Thus, whether the master or sub-program completes its task first, eventually a stage is reached at which the master program is informed that the sub-program has finished and is inactive. The master program may reactivate the sub-program immediately, or it may wish to do some preliminary data transfers before initiating the next set of output.

A more sophisticated programmer may improve on the utilization of the printer by dividing his working store into three areas, thus buffering the supply of data from master to
sub-program. By means of markers set within
the common area, the master and sub-program
can communicate their relative states, and need
only enter EXECUTIVE when it is necessary
for one or other to be suspended. However,
for most purposes the simpler approach will
produce a satisfactory increase in peripheral
utilization.

The remaining type of instruction which
causes entry to EXECUTIVE is the floating­
point group. These are macro-instructions
causing floating-point arithmetic operations to
be performed on operands by means of sub­
routines in EXECUTIVE.

Since a system may have a variable number
of peripherals and a variable number of pro­
grams, it is impossible to give absolute times
for any EXECUTIVE operations. However,
some sample times may give an indication of
typical EXECUTIVE operations. With a 2
microseconds core store, a peripheral transfer,
on a unit type of which there are 4 units in
the system, can be initiated within 350-600
microseconds. The time taken to suspend one
program and enter the next active program
can vary between 300-550 microseconds. The
size of EXECUTIVE varies with the configura­
tion of peripheral types and with the variety
of typewriter messages required by this par­
ticular system. The minimum size for EXECU­
TIVE, corresponding to a small system, is about
600 words. A system with an 8096 word core
store and 4 peripheral types might have a 1200
word EXECUTIVE.

Since EXECUTIVE has been written as a
set of routines which communicate with each
other by means of codeword addresses, the
EXECUTIVE required for a particular system
may be assembled quickly. Similarly, if a new
peripheral type is added to an installation, an
instruction which previously was illegal is now
routed to a new subroutine to perform the
required transfer. The subroutine is simply
added to the EXECUTIVE program at the end
of its area. Since EXECUTIVE may have
packages added or removed so easily, the indi­
vidual FP6000 installation may decide for itself
which facilities it wishes to have included at
the cost of some core store. For example, it
costs 40-50 words to permit ALTER messages
from the typewriter. In a system where an
established set of programs use most of the
operating time, it may be felt that the ALTER
facility is not as valuable as an extra 50 words
of core store available to the programs.

In this paper we have tried to show the way
in which the time-sharing facilities normally
only available on larger computer systems may
be realized on a medium-sized system by an
intermarriage of hardware and software for
minimum cost yet still retaining all the safe­
guards necessary for an operable system. We
have shown how the five criteria given by
Amdahl have been met and how, in addition,
we have been able to provide other facilities
designed to give a higher efficiency to the
system.

REFERENCE
1. GENE M. AMDAHL: New Concepts in Com­
puting System Design. Proc. IRE Vol. 50,
No. 5, May 1962.