APPLICATION OF THE PRINCIPLES OF TIME-SHARING IN THE DESIGN OF A MULTIPROGRAMME DIGITAL COMPUTER.

J.P.Penny B.Sc.

A Thesis Submitted to the Mathematics Department
of the University of Adelaide
for the Degree of Master of Science.

November 1960.

TABLE OF CONTENTS.

STATEMENT

This thesis contains no material submitted for any other degree in any University. To the best of my knowledge, due credit has been given in the text for the authorship of all material published previously by some other person.

17-11-60.

SUMMARY

This thesis is the result of a study undertaken in conjunction with the design of the digital computer CIRRUS, at present under construction in the Electrical Engineering Department of the University of Adelaide. The purpose of this particular study was to investigate the possibility of incorporating in the CIRRUS design facilities to allow multiprogramme operation on a time-sharing basis.

It is believed that multiprogramme operation would be highly desirable for CIRRUS. The chief grounds for this belief are that the efficiency of peripheral units should be considerably improved and that general operating conditions should be more satisfactory. For truly effective multiprogramme operation, the computer should differ considerably from a conventional computer. The characteristics of a multiprogramme computer and the effect which a computer of this form might have on various aspects of digital computing are discussed.

The most appropriate form for a multiprogramme operating system for CIRRUS is derived and the system which the author proposes should be used with CIRRUS is described in detail.

1. INTRODUCTION.

The development of the digital electronic computer has resulted in a machine which, unlike some earlier large scale calculating machines such as the Hollerith tabulator, is essentially capable of initiating only one simple arithmetic or transfer operation at a given time. In his expository paper (1958), Gill suggests three reasons for this:

(a) The large store size of the electronic computer makes it difficult to allow for comprehensive simultaneous transfers.

(b) Electronic techniques are less readily adaptable to simultaneous control of different channels than are mechanical devices.

(c) The spectacular operating speeds possible with electronic machines made other speed-increasing features, such as multiple operation, seem unwarranted.

Computer designers are continually searching for ways of improving the efficiency of computers. Certain aspects of computer development in the past few years have brought about a reappraisal of the possibilities of some kind of multiple operation. Most important of these has been the big discrepancy between the operating speed of the computer itself and the rate of transfer of data to or from the computer. Because of the essentially mechanical nature of

input-output equipment which limits its potential speed, this discrepancy has become more significant with each advance in electronic techniques which has pushed up computing speeds. The wide use of magnetic tape gave a big impetus to these new lines of thinking. The lengthy units of data as written on magnetic tape meant there was a much longer and more obviously usable period of time between the initiation and completion of each data transfer. In the time taken to read a data "block" from magnetic tape, the computer could have carried out a thousand or so arithmetic operations.

As a result, steps were taken so that data transfers and computation could occur simultaneously. Only a little of the computer's time was needed to initiate a data transfer, which then proceeded independently while the computer continued with the computation. The two processes were said to be "time-sharing" the computer. In these early applications of the principle of time-sharing, both operations were part of the same programme, and this entailed a fair amount of organisation by the programmer to achieve an efficient arrangement.

With increasing computing speed making the slowness of peripheral equipment more evident, other problems arose concurrently. Prominent amongst these was the difficulty involved in getting a human operator to use a super-speed

machine efficiently. With a computer faster by a hundred times, each second wasted by the operator represents the loss of a hundred times more potential work.

To assist in overcoming these problems, it was suggested that a computer could be time-shared by two or more distinct programmes. If a programme in operation were delayed for any reason, another could automatically make use of the time.

Many existing computers could, with relatively slight modification, carry out this "multiprogramme" operation to some extent. However, for truly efficient multiprogramme operation, the principles of time-sharing must be taken into consideration from the very first stages of a computer's design. The most important requirement for a multiprogramme computer is that, in its operation, there must be an additional, over-riding control, to determine how the computer's time is to be shared. This control can be brought about with hardware or with a master computer programme. Whatever form the controlling mechanism takes, it must be able in some way to keep a check on the data transfers to or from peripheral units.

The building of a new computer which differs radically in any way from conventional computers must result in a reassessment of current computer usage. In this respect, the multiprogramme computer will be no exception.

Its largest influence will be felt on present ideas in the design and use of peripheral equipment. By being able to use for computation most of the time needed for data transfers, the multiprogramme computer has less need of high-speed peripheral equipment. It can operate efficiently on jobs comprised largely of data input and output, thus obviating the necessity for off-line data converters. By working on another programme while an operator considers what he should do next, the need for efficiency on the part of the operator is lessened, and a new approach on methods of operating the computer is likely.

When the Electrical Engineering Department of the University of Adelaide decided to develop, with the assistance of the Mathematics Department, its own digital computer - to be called CIRRUS - it was felt that consideration should be given to the incorporation of facilities for multiprogramme operation. CIRRUS will be a relatively small computer, of moderate speed and low cost; its store capacity will be 4096 words, each of 36 bits, its addition speed about 45 μ secs., and its cost (for components only) between £10,000 and £15,000. A general specification for the proposed CIRRUS computer (which is projected for completion during the second half of 1961) is given in Appendix A.

Hitherto, most discussion on time-sharing has centred on its application to large, fast computers, chiefly because

the shortcomings of peripheral equipment are more obvious with faster computers. However, as a result of a study undertaken to assess the practicability of a multiprogramme system of operation for CIRRUS, the author believes that time-sharing would be particularly beneficial with a computer of this type. This thesis records the results of this study and includes a detailed specification for a multiprogramme system of operation which the author proposes should be used with CIRRUS. It is the author's view that incorporation of this system into the CIRRUS design will result in a computer considerably more efficient and versatile than would have otherwise have been the case.

2. THE CASE FOR A MULTIPROGRAMME COMPUTER.

The striking advances made in computer operating speeds over the past few years seem hardly to have kept pace with increasing realization of what a computer can do. The demand for time on any existing computer is usually heavy, regardless of what optimistic predictions may have been made before the computer was installed. For this reason, in addition to the measures adopted during the designing of the computer to improve its performance, the computer, when completed, is usually operated in a way calculated to maximise its output. The busy computer should be run with stream-lined efficiency; indeed, on many computers, the possible loss of a few seconds of computing time is regarded as a serious matter.

It is pertinent to ask to what extent this emphasis on time-saving is justified. To answer this question, and to gain some idea of the advances in speed of more recent computers, let us consider the average number of simple arithmetic operations which can be performed by various calculating machines. These are summarised in Table 2.1.

TABLE 2.1.

NUMBER OF ARITHMETIC OPERATIONS PERFORMED PER SECOND.*

CALCULATOR	CONSTRUCTED	OPERATIONS
Hollerith Tabulator (Punch card, electro-mechanical)	Pre 1950	2.5
CSIRAC (C.S.I.R.O. Automatic Computer)	1950	500
WREDAC (W.R.E. Digital Automatic Computer)	1955	2,500
CIRRUS	1960-61	13,200
I.B.M. 7090	1959	200,000

*Based on the average time to carry out a simple calculation of 9 additions and 1 multiplication.

It can be seen that the loss of even a few seconds of computing time on a modern computer means the loss of potential work which is far from negligible. Time on a big computer is also extremely costly. The I.B.M. 7090 is probably the fastest and most expensive computer generally available commercially; the cost for computing time on it is approximately £5 per minute.

Hence, it may be argued that computers have outgrown their operators. Where they have become so fast that using them efficiently is difficult, the possibility must be considered that more manageable speeds might prove to be

more economic. Perhaps the use of ten separate slow-speed computers rather than a single one ten times faster, might be a better proposition. However, it is a fact that the ratio of operating speed to cost for a computer of given store capacity becomes much more favourable as speed increases. Strachey (1959) considers that a computer with operating speed in the micro-second range costs perhaps fifty times more than one whose speed is in the milli-second range. He concludes that work could be done with the microsecond machine at about a twentieth of the cost with the millisecond machine. However this pre-supposes that the operating efficiencies of the two machines are comparable.

It is obviously desirable to consider ways of improving the operating efficiency of computers, when preparing the basic system design. Before doing this, let us examine the ways in which the computers of today are inefficiently used. The most serious source of time loss in many computers is the delay between the start of a data transfer to or from the computer and its conclusion. The input or output medium (usually punch cards, paper tape or magnetic tape) must be mechanically passed through the input or output unit so that the data transfer can take place. Operating speeds of mechanical devices are intrinsically limited; with paper tape, for example, the computer could

accept data at a rate perhaps a thousand times faster than the paper tape reader is capable of supplying it. How significant this discrepancy is between input and output rates and computer operating speeds may be understood from the figures given in Table 2.2., which shows the number of additions which can be performed in the time required for the input or output of a single number.

With a computer such as the I.B.M. 7090, punch cards would be used sparingly, most input or output being made with magnetic tape. Although magnetic tape is so much more efficient than the input-output media which preceded it, its advent into wide use provided the stimulus for the first serious attempts to use the waiting time involved. There were two important reasons for this. Firstly, magnetic tape data is transferred in blocks of several hundred characters, so that the redundant time is consolidated into more manageable units. Secondly, for a computer operating on binary numbers, information on magnetic tape is always in a binary form. The conversions between the decimal form necessary on paper tape or punch cards and the binary form required by the computer themselves consumed a fair proportion of the waiting time with the relatively slow pre-1955 computers.

Although probably the most significant source of lost computing time, the slowness of peripheral units is by no

TABLE 2.2.

A COMPARISON OF INPUT/OUTPUT RATES AGAINST ARITHMETIC OPERATING SPEEDS.

COMPUTER	MEDIUM CONSIDERED	TRANSFER TIME (ONE 10 DIGIT NO.)		ADDITION TIME (TWO 10 DIGIT NOS)	NO. OF ADDITIONS PER TRANSFER	
		INPUT	OUTPUT		INPUT	OUTPUT
Tabulator (pre 1950)	Punch Cards	50msec.	50msec.	400msec.	.13	.13
CSIRAC (1950)	Paper Tape	100msec.	400msec.	2msec.	50	200
WREDAC (1955)	Paper Tape Magnetic Tape*	56msec. 3.4msec.	400msec. 3.4msec.	250μsec.	220 14	1600 14
GIRRUS (1961)	Paper Tape	33msec.	330msec.	45μsec.	730	7300
IBM 7090 (1959)	Punch Cards Magnetic Tape	30msec. 96μsec.(min) 400μsec.(max)	75msec. 96μsec.(min) 400μsec.(max)	4.4μsec.	6800 22 91	17000 22 91

* The tape units considered are those originally supplied with WREDAC in 1955.

means the only source. The computer is also used inefficiently because it must be controlled to some extent by a human operator. An operator would probably waste several seconds in, say, selecting a paper tape and loading it in the reader. Since CIRRUS will be able to invert a 10 x 10 matrix in about 5 seconds, and the I.B.M. 7090 to invert it in less than a second, this time loss by the operator must be considered as serious. One result of this has been that "smaller" jobs - jobs which may be of the same order of size as those for which electronic computers were first intended - are often discouraged on large computers. Another effect has been a tendency to take as much responsibility as possible from the operator's shoulders by allowing the computer to control itself through elaborate "supervisory" programmes.

Programme testing is another activity during which a computer is used at well below its capacity. Probably the most popular method of programme testing, among programmers at least, has been for the programme to be run under manual control, the programmer deciding at each stage what steps should be taken next. In any such testing run, which could be of anything from a minute to half an hour in duration, it is unlikely that the computer would actually be operating for more than about five per cent of the time. On large computers, manual programme testing is usually

forbidden, programmes being checked by using diagnostic routines.

The amount of programme testing is usually particularly significant with a computer in a university, where the number and variety of programmes (and programmers) is likely to be much greater than on computers which are used largely for data processing. The report of the Melbourne University Computation Laboratory for 1959-60 (Hirst 1960) shows that 39.3% of the useful time of the computer was spent on testing programmes.

As a further example of wasted time, we may adduce the maintenance and adjustment of peripheral units. Most computers have a fair range of peripheral equipment and failure of a peripheral unit is probably a more common occurrence than failure of the computer itself. Putting the unit back into service requires the computer for short spells of time, often over a considerable period. It would be extremely useful if a practical way were found by which the computer could continue to work with other units while this adjustment was carried out.

To estimate with any accuracy how much time is lost through these causes is impossible; it depends on the computer, its staff, and the nature of the work on which it is engaged. Nevertheless it is evidently considerable.

It is logical to give to the computer itself the task

of utilising these small pieces of time for it is ideally equipped to handle a problem comprised largely of decisions which must be made at high speed. As mentioned earlier the first serious attempts to get a computer to do this were made in conjunction with magnetic tape transfers. In a simple form, this was parallel operation of data transfers with computation. The onus was on the programmer to request data from magnetic tape some time before it was required, and then to allow computation to continue until most of the data transfer time had been used up. The procedure was not highly efficient and not at all convenient. A big disadvantage was that fairly complex circuitry was necessary to make it possible for an entire block of data to be read into the computer without affecting computation.

Techniques to achieve a high degree of parallel operation of tape units and computation have tended to make tape units extremely costly. A desirable feature of any magnetic tape system is that the programmer should be able to direct that a particular block be found. When equipment to do this is included with the tape unit, the unit and its control hardware assume the proportions of a small computer.

The obvious desirability of some form of multiple simultaneous operation has influenced the entire system

design of some of the most recent computers. Often, the intention has been to provide a multiprogramme system of operation, or, as it is often called, "parallel programming". The basic principles on which a multiprogramme computer would operate may be briefly summarised:

(a) The computer will, where possible, store two or more distinct programmes.

(b) The computer control unit will share its time between these programmes, attempting to be occupied with a programme at all times.

The widely used term, "parallel programming", seems to imply actual simultaneous operation of different programmes. Except that peripheral transfers may occur with arithmetic operation on other programmes, actual simultaneous operation is rarely the case.* In fact, with most computers which are to have parallel programming, there will be a distinct priority system with some programmes taking precedence over others; the term "parallel" is to some extent inappropriate. The mode of operation must definitely be distinguished from that of the Hollerith tabulator which could initiate and perform transfers between several counters in a single machine cycle.

---

*The IBM STRETCH is to have actual parallel operation with the facility of working on up to seven (neighbouring) instructions simultaneously (Codd, et al, 1959). However the STRETCH, to cost $15m,. is really several computers combined.

An important variation of the multiprogramme idea is seen in the Minneapolis Honeywell 800 computer (Harper 1960). The usual mode of operation of this computer involves obeying a single order of each of up to eight programmes in turn. Advance scanning allows those programmes which cannot proceed for some reason to be omitted from this sequence without any loss of time. Provision is made for priority to be given to a more important programme when required.

At first glance, there may appear to be false economy in the multiprogramme idea. To store more than one programme in the computer seems to imply that the computer's store must be very much larger than would otherwise have been necessary. However, a computer must be large enough to cope with the largest problem which it is required to handle. Most problems will be very much smaller so that several of these could be stored together and operated on a time-sharing basis. There will, of course, always be an occasional programme whose demands for store capacity mean that it must operate on its own.

So far, in the limited amount of published work, the value of time-sharing a computer has been stressed mainly in the case of computers which are both large and fast (and therefore expensive). CIRRUS will certainly not fall into this category, but it is the author's view that multiprogramme operation along time-sharing lines is

equally suitable for a computer of CIRRUS' type. This view is based on several facts. The strongest argument for time-sharing lies in the inefficiency of peripheral equipment and, CIRRUS, which must rely largely on paper tape, will, as can be seen from Table 2.2., have a particular need for time-sharing. Although the nature of the work for which CIRRUS will be used will not involve much data-processing (with consequently less emphasis on input and output), there will certainly be a great variety of problems with operator's time and programme testing time being proportionately greater.

As a result of the considerations outlined in this section, the author feels it is reasonable to draw a conclusion, which may be stated:

CONCLUSION I.

Provided investigation shows the idea to be practicable, CIRRUS should be designed for multiprogramme operation on a time-sharing basis.

3. CHARACTERISTICS OF A MULTIPROGRAMME COMPUTER.

A conventional computer operates on a single programme, obeying its orders in sequence and pausing only when some prerequisite for an order is still to be fulfilled. On the other hand, a multiprogramme computer must store several programmes, systematically sharing its time between those programmes which are ready to continue. With such a fundamental difference in its mode of operation, a multiprogramme computer can be expected to differ considerably in internal structure and organisation. In this section we shall consider certain of these differences.

3.1. The Director.

The multiprogramme computer must inevitably be more complicated in its system of operation than a conventional computer. Because of this, an additional controlling mechanism must be introduced. The extra control required is made up largely of decisions which must be made according to the circumstances prevailing. For example, at any time, a decision may be necessary on which programmes can be followed and which of them is the most suitable. The computer order-code is well equipped for handling a problem of this nature. A programme for exerting control over a multiprogramme computer has been termed a "Director" (Strachey 1959).

3.1.1. The composition of the Director.

It is likely that a multiprogramme system of operation could be achieved by programme for many existing computers with only relatively minor modifications to the computer (such as for example, providing some means by which the Director programme could check the state of the peripheral units). With a computer not expressly designed for multiprogramme operation, such a system is unlikely to be very effective. One big disadvantage would be the using up by the Director of an appreciable portion of the computer's store. As the Director programme would be reasonably permanent, it should ideally be provided in fixed storage. Suitable types of fixed store are at present available which are both faster in operation and cheaper to build than variable store. The Director will however require a small quantity of variable store for working space. This should be inaccessible to programmes other than the Director to avoid interference.

It would also be possible to provide the functions of the Director either partly or wholly with additional electronic hardware. As an example, let us consider the "Data Scanner " to be included in the projected AEI 1010 computer (AEI 1960). In the AEI 1010, control of the peripheral units will be "centred in a separate control

unit", (the Data Scanner). Although this control unit will be simpler than the central computer control unit, its cost will still be appreciable. Most of its functions could be adequately performed by programme. This would mean in effect that the computer control unit was doing the work of this special control unit. An appreciable amount of hardware would be saved, but at the cost of central computer time which would otherwise have been available for computation. Any decision as to the best balance between programme and hardware must depend on the purpose and scope of the computer concerned.

Strictly speaking, a programmed Director held in fixed store as described earlier has been provided as hardware.* However, when distinguishing between operations carried out with hardware or with programme, it would be entirely inappropriate to class this as hardware. The language in which it is compiled or expressed is always that of the computer order code, rather than that of electronic circuitry.

3.1.2. The functioning of a Director.

Before attempting to prepare a specification for a Director, it is worth while to consider certain basic requirements;

(a) As far as possible, the programmer should

---

* But in a very cheap form.

not have to consider the fact that his programme will operate in conjunction with other programmes.

(b) The increased complexity of internal operation should not be reflected in increased operating difficulty.

The Director actually exerts control over the whole of computer operation. For this reason, when constructing the Director, there is an opportunity not merely to satisfy these stated requirements, but to include facilities to make the task of the programmer or operator easier than it would otherwise have been.

Where the Director is a programme, operation of the computer may reasonably be regarded as the continual operation of a single programme, the Director, with the various working programmes being merely ever-changing sub-programmes of this master programme. An example of this view of a Director is given by the flow-chart of the proposed CIRRUS Director, shown in Appendix B.

To understand the functioning of a Director, let us as an example consider what is perhaps the most obvious type of multiprogramme operation. A set of programmes held in the computer has a definite order of preference or "priority". The highest priority programme which is ready for operation is followed at all times. Rather

than picturing the Director as mentioned in the previous paragraph, let us take the view that a single programme is operated until, in response to certain circumstances, the Director takes control, subsequently either returning control to the same programme or to another. It is easier to consider the Director in this way when coming from established ideas of the operation of a conventional computer. Broadly these "certain circumstances" may be:

(a) The operator wishes some action to be taken. In this case, the Director must either carry out the operator's request, or, if it refers to a programme other than the one currently operating, the request can be recorded for implementation at a more appropriate time.

(b) The programme in operation cannot proceed further until a data transfer is completed, or until the operator takes some action. Here, the Director must select the next programme lower in priority which is ready to continue. The cause of the programme switch must be noted, so that control can be returned when conditions permit.

(c) A data transfer concludes, or an operator is ready for a programme to continue. Where the data transfer or delay by the operator has caused a switch from a programme as stated under (b) above, control must be returned to the programme in question.

(d) A programme ends. Here, the Director must select the highest priority programme amongst those which remain and are ready to proceed.

3.2. Peripheral Equipment.

The most obvious desirable requirement in peripheral equipment for a multiprogramme computer is that there should be a greater number of input/output units available for allocation amongst several programmes. This may not be absolutely essential however; for example a computer with a minimum amount of both paper and magnetic tape equipment can be usefully time-shared between two programmes, one using paper tape and the other magnetic tape. But, in practice, the number of programmes which can be run concurrently will be limited as often by the number of peripheral units as by the size of the store. These additional peripheral units should also be available for use by a single programme if required.

For computation to proceed during peripheral unit operation, one of two alternatives must hold. Either the unit must have access to the addressable store, independent of any circuitry used in the computation, or the unit must have its own buffer store. In providing direct access for each unit to a store which the computation is using, it may be difficult to avoid interference between the data transfer and the computation,

as the two processes are both concurrent and operating independently of each other. Where a buffer store is used, this is filled directly with data from the input medium, and when the buffer is filled the computer takes charge at a convenient time and transfers the buffer's contents to the store. The buffer acts in a similar way as an intermediate store during output.

The Director must be able to determine whether a unit is available for a transfer of data to be initiated. It must also be "informed", or at least be able to find out, whether any particular data transfer has been completed.

There are probably many ways of achieving this; one simple way is for each unit to have an "indicator" which is in one state while the unit has a transfer in progress, and another state when no transfer is taking place. Whether Director action is necessary may be determined from the actual state of an indicator, for example, it may show that the unit is unavailable, or the need may become apparent with the change of state of an indicator, showing that a unit has become available.

The term "indicator" implies subservience to a Director which makes its examinations at times suitable to itself. Alternatively there could be cases where it is more appropriate for the peripheral unit to "interrupt" the Director. Before discussing this further, let us

consider an historical example of an elementary form of time-sharing. This was a procedure used on EDSAC (Wilkes and Willis 1956) whereby a magnetic tape unit could be run backwards or forwards under computer control for a given number of blocks while most of the time was available for computation.

A special marker on the tape with each block of data set a flip-flop as an indicator when it passed the reading head. A computer order was available which examined this indicator, resetting, and passing to the next order if it had been set, or jumping if the indicator had not been set. A flow-chart of the procedure is given in Fig. 3.1. It was necessary to include the special order at intervals throughout the computation programme. Although it was no doubt inconvenient, there was a definite advantage in having the indicator checking order at frequent intervals. If the check were made soon enough after the setting of the indicator for the tape to be still moving, it was then not necessary for the tape to stop completely unless the required point had been reached.

If it had been possible also to read data while the computation continued, a slight variation of this procedure could have produced an efficient block searching routine. However, it would have been necessary for each block to be read entirely without recourse to the

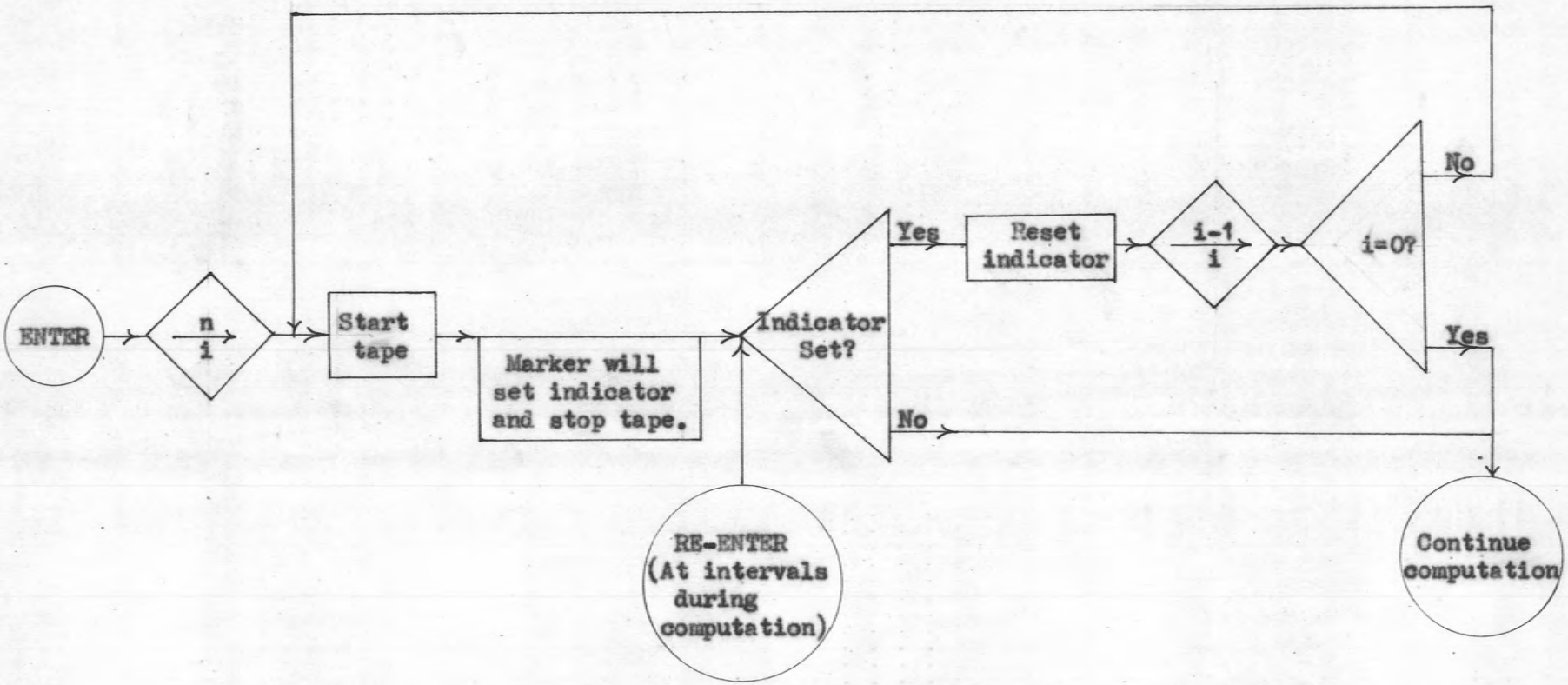


Figure 3.1.:- Diagram showing the EDSAC procedure for moving a magnetic tape for n blocks.

computer, as the tape could not delay in mid-block to wait for an indicator checking order.

If we wish to have only a small amount of magnetic tape buffering, with the computer ceasing computation at frequent intervals to carry out the assembly of the data block in the store, the signal that the buffer has been filled must break into or "interrupt" the computation with a minimum of delay. Strachey (1959) suggests a method of time-sharing the computer during magnetic-tape transfers, applicable only to a super-speed computer and requiring no buffering and an absolute minimum of other hardware as even the assembly of characters into a word is done by the computer. Assuming a character appears every 30μsec., and must be accepted within, say, 10μsecs. of its appearance, a superspeed computer would be able to bring into operation a word assembly and storage programme. However the computer would have to be really fast. If only two tape units were required to be able to work together, the whole process of switching programme, word assembly and store transfer would have to take less than 5μsec. to allow for the possibility of characters appearing from both tape units at the same instant. With the system quoted in this example only a third of the computer's time would be tied up in controlling the data transfers, and the rest would be

available for other work.

Using an elementary buffer for each tape unit which would hold a complete word, (or preferably two single word buffers between which the unit can alternate) would make timing much less critical. However the "interruption" requirement is still quite definite, as the tape unit cannot stop between single words. If the computer must be prepared for interruptions with limited waiting periods, the programme for handling the interruptions cannot rely on using any sections of the computer unless it is certain that they will be available at regular and short intervals.

Let us assume there are n independent interruptions (not necessarily all associated with peripheral units), which are handled in a priority order based on their maximum waiting times,

t_i, $i = 1,2.....n$ with $t_m \leqslant t_n$ for $m<n$

Assume that the computer will require a time T_i to pass the critical stage for the interruption, and a time T_i' ($T_i' \geqslant T_i$) to deal completely with the interruption (including switching time).

The inequality

$$T_i + \sum_{j=1}^{i-1} T_j' < t_i \qquad .....3.1.$$

must hold for all i, in case all interruptions occur together.

Since we must assume that some sections of the computer will be available to deal with these interruptions, we must allow for the maximum possible delay before any required section becomes available. If this delay is τ, then

$$\tau < t_i - \left[\sum_{j=1}^{i-1} T_j' + T_i\right] \qquad \text{.....3.2.}$$

must hold for all i.

In view of the practical difficulties involved at this stage of computer development, producing a system along the lines suggested by Strachey which allowed for a reasonable number of simultaneous magnetic tape processes would not be easy.

We may summarise discussion on buffering of peripheral units with a multiprogramme computer by saying:

(a) In general, some buffering is necessary for each unit.

(b) The amount of buffering required may possibly be quite small.

The extent to which buffering will actually be necessary is a function of :

(a) The rate at which the unit supplies data.

(b) The maximum possible delay between the instant at which the buffer is ready to be emptied, and

the time the computer is ready to begin emptying it.

(c) The time which the computer takes to empty the buffer.

(d) The number of units which it is desired should be able to run simultaneously.

Relating this to Equations 3.1, 3.2, we can say:

(a) t_i is the delay between the filling of the buffer, and the time when it must be ready to receive more data.

(b) τ is the maximum delay before the computer is ready.

(c) T_i' is the time which the computer requires to empty the whole buffer; if the buffer is emptied in sections (characters, half-words, words etc.) T_i is then the time required before the first section is emptied.

(d) n is the number of units which are to run concurrently.

Similar factors are also involved when considering simultaneous writing from several tape units, or simultaneous reading and writing.

Each paper tape unit commonly possesses its own single character buffer. In a paper tape reader, this buffer will take about 2 to 5 msec. to fill, but, once assembled, the character can be transferred to the computer store in a matter of a few microseconds. Although

the computer usually must do much more with each paper tape character (e.g. convert between decimal and binary), than with a magnetic tape character, the margin of time to be saved is much greater. In addition, there is not the problem of interruption, as most common paper tape units will wait indefinitely between each character (t_i of formulae 3.1, 3.2. is infinite). However, to make efficient use of the few spare milliseconds available with each character, the switch between the computation programme and the character handling programme must still be quite rapid..

To conclude, it seems reasonable that, with several different input and output units being shared between different programmes, there should be more than one operating console. Although one operator might be able to run several programmes by himself, more effective use of a multiprogramme computer will certainly result if several operators can have simultaneous access to it.

3.3. Multiplicity of Auxiliary Registers.

In theory all that any programme need have exclusively to itself in a multiprogramme computer is adequate programme and working space and the required number of peripheral units. Auxiliary registers, such as accumulators and B lines can be time-shared by the programmes, provided that at the time of switching to a

different programme, nothing of importance is held in these common registers. This can be assured either by allowing programme changes only at times when these registers are known to be empty, or by storing their contents in the programme's working space when the programme is suspended.

The first alternative is fairly restrictive, and the second inconvenient and time-consuming. It is far better to be able to allocate separate accumulators and B lines to each programme. Once again, the ideal is costly, but an inexpensive compromise arrangement which has been used in the system design of CIRRUS is worth considering. This is shown in simple outline in Figure 3.2.

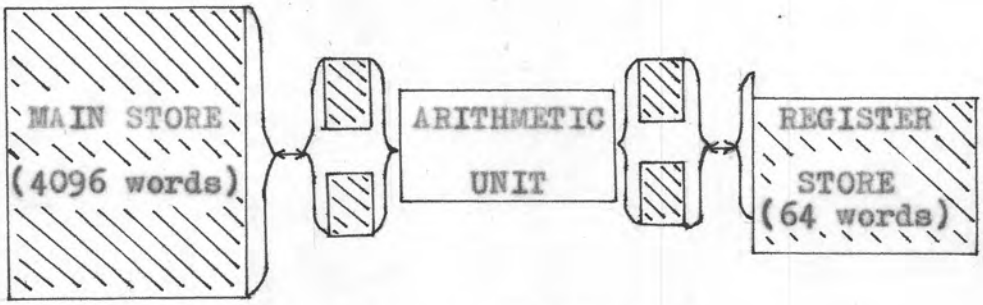


FIGURE 3.2.

SYSTEM STRUCTURE OF CIRRUS (SIMPLIFIED).

The main and register stores are core stores, the required number of addresses being allocated to each programme as it is assembled. As far as the machine order code is

concerned the register store locations act either as accumulators or B lines, and there can be several of each for every programme. If only the working registers (and arithmetic unit) are time-shared by different programmes, a programme switch can occur at any time that the working registers are not holding wanted information. This means each programme may have several accumulators and B lines without any necessity for storing their contents elsewhere when the programme is changed.

4. THE INFLUENCE OF THE CONCEPT OF MULTIPROGRAMME OPERATION ON VARIOUS ASPECTS OF DIGITAL COMPUTING.

The multiprogramme computer will have a wide effect on digital computer techniques. Only experience in operating such a computer will show the full extent of this effect, but several obvious consequences can be predicted.

4.1. Peripheral Equipment and Data Converters.

The need for a review of ideas on buffering of input-output equipment is apparent from what has been discussed in Section 3.2; however, this is only one of the ways in which multiprogramming will affect conventional peripheral equipment. One important change will be the lessening in emphasis on speed of some peripheral devices. If all the spare time from data transfers from the device can be used on some other programme, then the amount of this time, or, in effect, the speed of the unit, is less significant. This may not be the case however with magnetic tape in certain commercial applications. Here, the amount of computation may be so small that the magnetic tape speed is the only factor limiting the rate at which the job can be done.

However, let us consider a paper tape reader where only 10% of the waiting time for a character is absorbed

by handling of the previous character with the remaining 90% being usefully employed by other programmes. A reader 10 times faster would be advantageous only in that it would reduce the probability of all programmes being unable to continue because of incomplete data transfers.

Paper tape reader troubles were common in the author's experience with WREDAC and apparently this has also been the case with SILLIAC.* However, a reader of identical make to those used with WREDAC and SILLIAC has been used very satisfactorily on CSIRAC. One difference is that, on CSIRAC, because the computer is not capable of dealing quickly with characters, the reader operates at only half the speed of those on WREDAC and SILLIAC. It appears that using peripheral units at speeds which have had detrimental effects on their reliability of operation, has been a common practice in the past.

As a corollary, it seems likely that the multiprogramme computer will cause the return to favour of certain inherently slow peripheral devices. Plotters or teleprinters, both of which have distinct advantages but are limited in speed, may again become popular. Very high speed line printers are now sometimes used for direct output from the computer; it is likely that reasonably slow line printers could be effectively used with a multi-

---

* The Sydney University version of the Illinois computer. This information was received in a personal discussion with B.Swire.

programme computer.

Off-line data converters are unnecessary with a multiprogramme computer. The cost of the WREDAC output converter used solely for displaying in graphical or tabular form the information written by the computer onto magnetic tape, was more than 25% of the cost of the computer itself. A time-shared multiprogramme computer can, with the expenditure of only a small amount of its time, be used as a data converter of any kind. Acting as an output display converter, for example, the computer will do the job perhaps more effectively than a specialised converter as all the logical functions of the computer are available to help arrange the display. The instructions for the display will be held on a short piece of programme tape or pack of punched cards; the WREDAC output converter had to be "programmed" with a plug-board, two plugs being required for each decimal digit. Similarly, paper tape or punch card to magnetic tape converters may also become redundant because the computer itself can be used.

4.2. Operating Techniques.

The operating techniques found to be most suitable for a multiprogramme computer will differ considerably from those in use with a conventional computer. These differences will usually be advantageous, but not

exclusively so. One unfortunate complication arises from the fact that, if the operator wishes to intervene in any programme, the Director must be able to identify the programme to which he refers. One way in which this could be done would be to include separate operating positions, and restrict each position to a single programme.

However, this necessity for programme identification is a relatively small price to pay compared with some of the operating advantages of the multiprogramme computer. The operator can assume that the time he spends at the console preparing his work will be usefully employed by other programmes, and he can take more care and make fewer mistakes.

Methods used by operators in programme testing will need reviewing. There is a possibility that the once popular, but discredited method of operating the programme under test with a high degree of manual control will become widely used again. The Director should be able to merge the requirements of a programme under test with its other work-load. However, programme testing presents two problems with a multiprogramme computer. In the first place the computer cannot afford to wait on any single order while the operator laboriously decodes it from a display on a cathode-ray tube. Secondly, the possibility of an untested programme "running wild", and overwriting

other programmes, must be guarded against.

The answer to the first point probably lies in using a simple routine which will allow orders from any programme to be displayed on a monitor printer if requested. Controlling the printer for this operation will occupy the time-shared computer for only a small fraction of the actual printing time.

To prevent overwriting, some safeguards must be included. These may be done by "locking-out" all sections of store not allotted to the programme in question, or by checking the addresses in orders before they are obeyed to make sure they lie within the programme's bounds.

The first of these alternatives is expected to be provided (in hardware) with the projected Orion computer, a multiprogramme machine to be constructed by Ferranti Ltd. (Ferranti 1960). As a result of discussion with representatives of this firm, it is understood that this procedure of locking out the store sections of other programmes, will be carried out on all programmes, whether tested or not. This is a safety measure, however, to help in introducing the Orion with its unfamiliar multiprogramme structure to the highly competitive commercial market.

Checking addresses presents the difficulty that, whereas in most orders the addresses refer to store

addresses, with some orders (e.g. shift, input or output), this is not so. Orders such as these must be separately considered.

The lessening of emphasis on the quickness of the operator is particularly attractive with a scientific computer. It becomes practicable to have a certain amount of user participation in a computation. To illustrate how useful this might be, let us consider a numerical problem which for some reason cannot be solved with any programme in the library. This could be, for example, the solution of a set of simultaneous linear equations, the set being perhaps too ill-conditioned for solution with any existing programme. A programme to solve this problem would have two distinct parts:

(a) The substitution of estimated values to calculate residuals.

(b) The calculation of a further set of estimated values.

For the first part, the programme would be simple to compile and would save much manual computation. On the other hand, in adding to the programme the calculation of further estimates it may be quite difficult to avoid the possibility of getting a divergent sequence. This calculation would also replace only a small amount of intuitive effort by the operator in applying relaxation methods.

Using a time-shared computer with more than one operating station, only the first part of the programme need be provided. The computer will calculate the residuals and the operator can calculate better estimates, which he will then return to the computer. Provided more than one operating station is available, a technique of this kind should be valuable for many types of iterative process.

4.3. Automatic Process Control.

The controlling of industrial processes is another application of the electronic computer which is now assuming considerable importance. An example is given by Guidero (1960), who describes the way in which a digital computer is used to control an electric power station.

In process control, the computer accepts readings from various instruments monitoring the process either at regular intervals or in response to some alarm signals. This data is operated on with a computer programme, most appropriately a fixed programme, and the results either displayed for the operator, or used by the computer itself as a basis for decision on what action is necessary for further control of the process. In the example which Guidero describes the computer actually implements the action which it decides is necessary without recourse to the human operator.

There is a basic disadvantage in using a conventional computer in certain forms of process control. The decisions which the computer must make should most appropriately be made with a minimum of delay. This means that the computer should be reasonably fast and hence that all the computer's control functions will occupy only a small proportion of its total capacity for work. Cook (1960) describes how a small time-shared computer, the Elliott 802, is used for process control. For most of its time the computer can operate on base-load programmes, transferring control to fixed control programmes if a need for action arises. A request for action appears as a changed digit in a control word. Each programme must allow for examination of this control word to be made at suitable times - suitable in that the contents of the accumulator are no longer required by the base-load programme. Detection of a changed digit in this control word results in the correct control programme being brought into operation.

The multiprogramme computer is particularly suitable for this application, as it is a computer expressly designed to be able to transfer control from one programme to another at short notice. In addition the multi-programme computer will normally already have a fixed programme store for holding the Director programme.

This can readily be extended to hold the control programmes.

The problems involved in making the multiprogramme computer operate these control programmes in response to random interruptions from processes under computer control are similar to those involved in making it possible for the computer to be operated from several completely independent operating stations. Provided the transmission of data can be handled satisfactorily, in the future the ideal way of supplying computing facilities for several separate laboratories may be to have a single large multiprogramme computer with an operating console in each laboratory.

5. THE MULTIPROGRAMME SYSTEM SPECIFICATION FOR CIRRUS - PRELIMINARY CONSIDERATIONS.

5.1. The Basic Form of the system.

The decision to make CIRRUS a multiprogramme computer leaves much unresolved. Construction of the Director will allow scope for providing a system suited both to the general specification of the computer and also to the way in which it is expected to be used. CIRRUS is intended largely for scientific computation. Its characteristics have been determined by what was felt would be most suited to the needs of an Australian University or an organisation such as the C.S.I.R.O.

Scientific computation requirements result in a work-load of different character from that of a computer used in commercial work. The computer in a university, in particular, can be expected to perform a great variety and number of problems. The staff attached to the computer is usually small, most jobs being programmed (and often operated) by the person directly interested in the project. Many programmes will be prepared once for a specific job, and never used again, and, for this reason, programme compilers are extensively used with a scientific computer.

It can be assumed that the daily work-load for CIRRUS will be both varied and unpredictable. The question of

scheduling a multiprogramme computer or attempting to run programmes in the most efficient combinations which has been discussed both by Strachey (1959) and Codd (1960), presents difficulties in this case. Strachey suggests that a "base-load" programme, one which is long running and requires little time for data input or output, should be run in conjunction with short duration programmes, programmes under test and so on, so that one programme is always available to "fill in the gaps between other programmes". Codd suggests a more elaborate procedure to be applied with STRETCH and which will enable the computer to assess the characteristics of all programmes to be run, and to choose them in "compatible" sets. However, with the work for which CIRRUS is to be used, both these ideas are difficult to put into practice. The Strachey "ideal" can be used, where practicable. The Codd procedure requires the programmer to provide the computer with knowledge of the programme's requirements both in space and time. To calculate the space requirement is easy; a special computer programme can do this in a few seconds. Calculating the requirement in time is not so simple. If the programme is to run only a few times on the computer the procedure is of more trouble than value, and if the programme is to be run repeatedly its time requirements may need reassessment on every occa-

sion, depending on the amount of data involved. The particular nature of the work for which CIRRUS is expected to be used will make any worthwhile scheduling difficult.

In the institutions in which a computer such as CIRRUS would be used the use of the computer for training future users, or, in a university, for training students as part of their courses in numerical methods or electronic techniques should be fairly common. With computer time a fairly costly item, allowing casual operators to gain practical experience has always been discouraged. Nevertheless, the 1959-60 CSIRAC report (Hirst 1960) shows that 2% of the total useful computer time was used for teaching. With multiprogramme operation, where the practising operator will be using the computer for only a fraction of the time which he spends on it, an appreciable amount of teaching will no doubt be carried out.

It is also worth noting at this stage that the class of job mentioned at the end of Section 4.2, that is, those in which the operator himself may desire to participate in the computation, will be fairly frequent.

Let us now make a summary of the important aspects of the probable CIRRUS work-load, which will influence the choice of the basic multiprogramme specification.

(a) The work-load will vary greatly from day to day.

(b) The operators will often not be highly skilled.

(c) Programme testing, use of the computer for teaching, and so on will be frequent - that is, there will be many occasions on which the demand for the central processor's time will be small.

It is now worth reiterating the two important requirements stated in connection with the Director in Section 3.1., viz,

(a) As far as possible, the programmer should not have to consider that his programme will operate in conjunction with other programmes.

(b) The increased complexity of internal operation should not be reflected in increased operating difficulty.

In view of all these points, the author suggests that a desirable form for the multiprogramme CIRRUS computer would be such that it behaves almost as if it were several separate and independent computers, but with the proviso that it must be possible for a single programme to make use of any or all of the facilities.

Let us state this as the second conclusion.

CONCLUSION 2.

The basic form of the CIRRUS multiprogramme computer should, as far as is convenient, cause it to

behave as a set of separate computers for the set of programmes to be run.

We cannot, of course, have the best of all possible worlds. If more than one CIRRUS computer is constructed variation of the multiprogramme system may be desirable, to suit a different case. In addition, experience in using CIRRUS will certainly dictate ways in which the system can be improved. We must therefore state a third conclusion.

CONCLUSION 3.

The multiprogramme system must be basically sound and flexible enough to allow for changes to be made in the light of experience or changed conditions.

5.2. The Influence of the CIRRUS Design on its Multiprogramme Specification.

The CIRRUS project is certainly not intended solely to provide a vehicle for the testing of time-sharing ideas; the computer design has its own individual characteristics which, in their turn, have had bearing on the multiprogramme specification. Those features of CIRRUS which are most important are:

(a) A wired, fixed-store is provided.

(b) The computer is micro-ordered - that is, the orders which the computer will obey form a small set of elementary functions, and the actual order code is

built up from these micro-orders.

(c) The chief medium for input and output is expected to be paper-tape.

(d) The computer is required to be of low-cost.

The wired store provides low-cost rapid-access storage.* This is therefore ideal for storing a Director programme. Using wired store, however, presents its own particular problems, the most important being the difficulty of incorporating parameters into the fixed routines.

The micro-order structure makes two levels of orders available for the knowledgeable programmer. The micro-order code is restricted but fast (from 1 μ sec. for a control transfer order to 7 μ sec. for an order involving the variable store). It is desirable, therefore, to consider using the micro-order code to provide those parts of the Director to which reference is most frequently made.

The supposition that paper tape will be the most commonly used input-output medium will have considerable bearing on the details of the multiprogramme scheme, but the possibility of the inclusion of magnetic tape should also be kept in mind.

---

* The cost per 36-bit word is expected to be about one tenth of the cost per word of variable store. The access time will be about one sixth of that for variable store.

The requirement of low-cost means that very little extra hardware can be allowed in achieving the multi-programme operation. This places most of the burden for Director functions on programme which is a cheap commodity. The only appreciable expense which must be debited to the multiprogramme system is the provision of extra peripheral units. Since time-sharing makes it possible for units to be slower and simpler than would otherwise be the case, this extra cost will be lessened.

This final point is quite important, so let us state it as the fourth conclusion.

CONCLUSION 4.

The provision of multiprogramme facilities must not materially affect the status of CIRRUS as a low-cost computer.

6. THE CIRRUS MULTIPROGRAMME SYSTEM SPECIFICATION.

In this section, we shall discuss the means by which multiprogramme operation will be achieved on CIRRUS. The multiprogramme system must be such that the stipulation of Conclusion 2, that is, that the computer should behave as several different computers, is complied with. It must also not conflict with the requirement expressed as Conclusion 3, that the system must allow for amendment and expansion, nor with the requirement stated as Conclusion 4, that it must not materially add to CIRRUS' cost.

We shall first, in Section 6.1., deal with the subject of programme switching, enumerating those circumstances which will cause a switch from one programme to another, and outlining the mechanics of the switching procedure itself. In Section 6.2., the way in which the operator will control the computer will be described. Having thus set out the basic principles on which the computer will operate, we shall finally in Section 6.3., examine in detail the operation of a set of programmes.

Throughout this section it is assumed that the general specification of CIRRUS, described in Appendix A, is known.

Conventions:

(a) Notation

To represent the functions of the arithmetic unit of CIRRUS, the symbols used are:

$\frown$ (and)

$\smile$ (or)

$\oplus$ (logical differ)

e.g. If A = 110, B = 101, then

$A \frown B \longrightarrow B'$ means that B = 100 finally

$A \smile B \longrightarrow B'$ " " B = 111 "

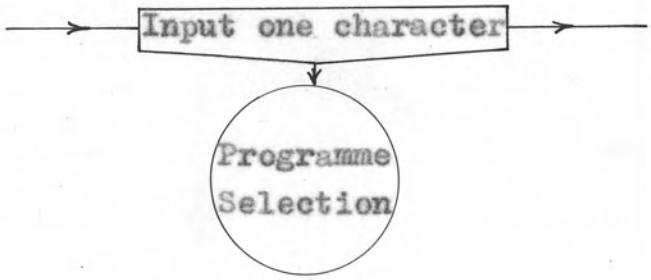
$A \oplus B \longrightarrow B'$ " " B = 011 "

In this example, A and B are undefined binary numbers. In later discussion, the quantities may be store addresses, the contents of store addresses, or any other quantity which may be defined. In context, there should be no ambiguity.

(b) Flow Diagrams.

The conventions adopted for compiling flow diagrams are largely those set out in the Handbook of Automatic Computation and Control, Volume 2 (Grabbe, et al 1959).

The notation



is also used with input or output orders to show that a programme switch will be made if the unit is unavailable,

(c) Peripheral Units.

In accordance with the plan to provide, apparently, several different computers, it will be necessary to allow for more than one operator. For the sake of discussion, the console layout shown in Figure 6.1., and based on

3 readers.

3 punches.

2 input/output writers.

4 magnetic tape units.

1 line printer.

is proposed.

In practice, it is improbable that the composition of the computer's peripheral equipment will be identical with this. In particular, it is likely that magnetic tape will not be included with the computer when it is first constructed.

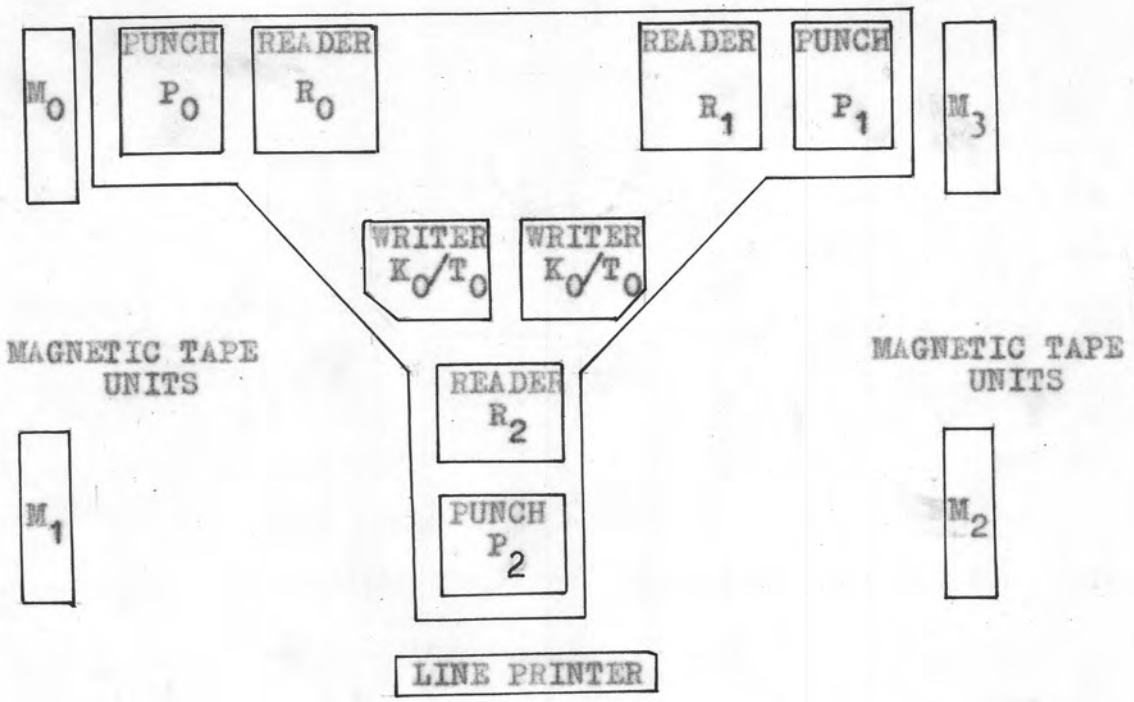


Figure 6.1.:- A suggested console layout for CIRRUS.

6.1. Switching Programmes and the Use of Indicators

In introducing the subject of programme switching, let us first consider the conditions which will cause a break in the operation of a particular programme. These conditions are shown in Table 6.1.

Of the situations listed in Table 6.1., numbers 3 and 6 arise within the operating programme itself. In these cases, transfer of control to the Director can be effected as part of the programme, and this will ensure that the control transfer occurs at an appropiate time. By an "appropriate time" is meant a time when nothing required by the current programme is held in any time-shared sec-

tion of the computer.

TABLE 6.1.

DIRECTOR ACTION ON CIRRUS.

CAUSE OF DIRECTOR ACTION	RESULT OF DIRECTOR ACTION
1. The operator wishes to intervene.	Control will be given to a special programme.
2. The operator is not ready to continue.	A switch to another programme.
3. The programme requires a peripheral unit which is not available.	A switch to another programme.
4. A peripheral unit completes a transfer.	If this unit is required by a higher priority* programme control will go to this programme.
5. A request is received from an external device which needs computer attention.	Control will be given to a special programme.
6. A programme ends.	Control will be given to a special programme.

*As far as priority is concerned, it is sufficient to say at this stage that there will be a definite order of precedence among programmes.

With cases 1, 2, 4 and 5 in Table 6,1., the condition for Director action arises independently of what is taking place in the programme in operation. In these cases, any switch to another programme must be withheld until the time-shared sections are free.

Figure 3.2., is a simplified diagram of the overall structure of CIRRUS. Let us accept at this stage the general principle that each programme shall have allotted to it its required amount of storage in both the main and register stores for programme and working space. The working registers (and the arithmetic unit) will be time-shared between the programmes; thus a programme switch may occur only when these are clear. The order structure of CIRRUS, with each machine order being built up of micro-orders, allows for the use of these registers within each machine order for holding intermediate values either in the computation or in inter- or intra-store transfers. At the end of each machine order, all values required for further use are placed in either the main or register stores. Hence, we can allow a programme switch at the end of any machine order.

Restricting programme switches to the point between orders means that a switch may be delayed for a period as long as the time needed to obey the slowest machine order.

6.1.1. Primary and Secondary Indicators.

Having listed those conditions which can cause a programme switch and decided when a switch can take place, let us now consider how these conditions are to be monitored.

Definition: The Primary Indicators, I_i with i = 1,2..18

A set of up to 18 independent flip-flops. Their setting and resetting will be independent of the computer. It will be possible to bring the "state of the Primary Indicators", I, into the arithmetic unit as an 18 bit half-word.

Each I_i for i = 2,3,.....18 will show the state of a peripheral unit, being zero if the corresponding unit is available for a transfer, and one otherwise.

The state of any individual indicator, $I_2 - I_{18}$, is the result of a multiple "or" circuit, as the availability of a unit depends on several conditions.

e.g. (a) I_5 is the indicator for a reader.

= 1, if the power is off.

OR if there is no tape in the reader.

OR if the manual switch* on the reader is "off".

OR if there has been a parity** failure.

OR if no character has been assembled in the internal buffer of the reader - that is, a character has been taken from the buffer and the reader is still in motion.

---

*The manual switch allows the operator to "lock-out" the reader during loading.

**Parity checking will be done within the reader itself as the character is assembled in the buffer. Unless the check is successful, the indicator will remain in the "unavailable" position. A light on the reader will show the operator that there has been a parity failure. The operator must pull the tape back ready to re-read the

(b) I_9 is the indicator for a punch.

= 1 if the punch is switched off.

OR if the tape has nearly run out.

OR if the character in the buffer is being punched.

I_1 monitors a second set of indicators, the "Secondary Indicators".

i.e. I_1 = 0 if all Secondary Indicators are 0

= 1 otherwise.

Definition: The Secondary Indicators, SI_i, i = 1,2...18

A set of independent flip-flops. The setting of these will be independent of the computer but the computer will be able to reset them individually. The Director can inspect the "state of the Secondary Indicators", SI as an 18 bit half word, or as two separate 18 bit half words if the number of indicators is greater than 18. Table 6.2., shows the proposed distribution of Primary and Secondary Indicators.

The Secondary Indicators will show the need for Director action from some cause other then the availability or otherwise of the peripheral units. Each Secondary Indicator will normally be zero, the need for Director action

---

offending character, and reset the reader, probably by moving the manual switch to the "off" and then back to the "on" position.

TABLE 6.2.

TENTATIVE ALLOCATION OF INDICATORS

	1.: PRIMARY INDICATORS	
I_1	Monitor for Secondary Indicators	
I_2, I_3	Input/output writers-keyboards	K_0, K_1
I_4 to I_6	Paper tape readers	R_0, R_1, R_2
I_7, I_8	Input/output writers-typewriters	T_0, T_1
I_9 to I_{11}	Paper tape punches	P_0, P_1, P_2
I_{12} to I_{15}	Magnetic tape units	M_0, M_1, M_2, M_3
I_{16}	Line printer	LP
I_{17}, I_{18}	Spare	
	2.: SECONDARY INDICATORS	
SI_1	"Buffer filled" signal from first half of input buffer of M_0	
SI_2	"Buffer filled" signal from second half of input buffer of M_0	
SI_3	"Buffer emptied" signal from first half of output buffer of M_0	
SI_4	"Buffer emptied" signal from second half of output buffer of M_0	
SI_5 to SI_8	Buffer signals for M_1	
SI_9 to SI_{12}	Buffer signals for M_2	
SI_{13} to SI_{16}	Buffer signals for M_3	
SI_{17}	Operating instruction request-keyboard 1	
SI_{18}	Operating instruction request-keyboard 2	
SI_{19} etc.	Automatic control requests	

being shown by a change to one in the relevant indicator.

Let us now examine the ways in which the Director will use these indicators. To decide whether a peripheral unit is available for a data transfer, the corresponding Primary Indicator will be examined (at the beginning of the execute phase of the transfer order), before the transfer is initiated. As the examination will be included in the execution of the order, the programmer will not need to consider it. If this indicator is 1, another programme must be selected; this is case 3 in Table 6.1.

When an operator is not ready to continue, (this is case 2 of Table 6.1.), this will, with certain exceptions which will become evident later, be apparent to the Director as an unavailable peripheral unit, because, for example, the manual switch of the reader may be "off".

The wish of the operator to intervene (Case 1 of Table 6.1.) and a request from an external device (Case 5) will each appear as a change from 0 to 1 in a Secondary Indicator, and, hence a change in the first Primary Indicator. The conclusion of a peripheral transfer (Case 4), will appear as a change from 1 to 0 in the corresponding Primary Indicator.

Definition: Expected State of Primary Indicators, EI_i,

i = 1,2....18. An 18 bit half-word, to be held in the upper half of register zero. The first bit will always be zero, as we expect all Secondary Indicators to be zero. EI_i for i = 2,3.....18 will be 0 or 1, depending on the state of the peripheral unit at the last Director intervention.

To detect changes in Primary Indicators, a comparison of EI and I will be made at the beginning of each machine order, that is, at a time when a programme switch is permissible. The routine phase of each order is then as shown in Appendix A. It is likely that this comparison can be made concurrently with the operation of bringing from the store the contents of the sequence counter. Therefore, provided EI and I are identical, no computer time will be lost.

The first bit of EI will always be 0, so that if any Secondary Indicator has changed, EI and I will differ in the first bit position. Corresponding to each Secondary Indicator there will be a micro-ordered programme to carry out the action shown to be necessary by the indicator change. These micro-ordered programmes will all be quite short; those corresponding to the operating instruction requests, for example, will merely set up a machine-ordered programme which can read a complete operating instruction while time-sharing the computer with other programmes. Similarly, programmes for dealing with

requests from processes under automatic computer control will probably be machine-ordered programmes brought into operation in this way.

Examples on the use of indicators.

Example 6.1.:

Checking whether a peripheral unit is available for a transfer. If unavailable the Director must record this, and then switch to another programme.

Definitions:

(a) Unavailable Peripheral Units, UP.

A half-word, held in the Director's working space, showing what units have been requested and unavailable. Unavailability is shown by the bit corresponding to the indicator being equal to 1.

(b) Requested Unit RU.

A half-word with one non-zero bit, taken from a table in the fixed store and showing the indicator corresponding to the peripheral unit.

Let us assume the unit in question is a reader.

The sequence of operations is shown in Figure 6.2.

Example 6.2.: Detection of the ending of data transfers.

If the unit concerned is required on a higher priority programme than the current one, control will go to the higher priority programme. The sequence

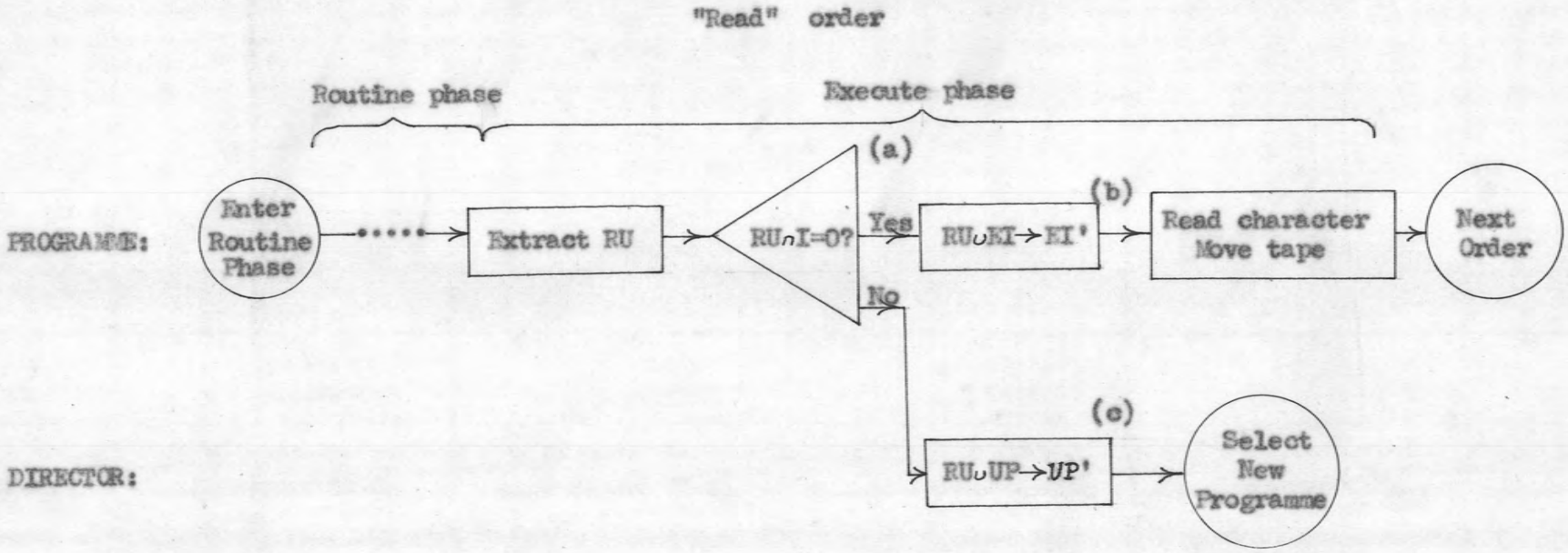


Figure 6.2.:- Diagram of the sequences of operations dependendent on the availability or unavailability of a peripheral unit.

Notes on Figure 6.2.:- (a) The question is: "Is the unit available?".

(b) The indicator is now expected to be 1.

(c) UP is amended to show that this unit is wanted when available.

of operations is shown in Figure 6.3.

Example 6.3.: Dealing with interruptions registered as a change in a secondary indicator.

The sequence of operations is shown in Figure 6.4.

6.1.2. Programme Selection - The Sequence Counter Ladder.

To change programmes on CIRRUS will be a simple matter. The first step in obeying any machine order involves bringing the address of the order from the programme's sequence counter. As each programme will have its own sequence counter showing the next order of that programme we need merely change from one sequence counter to another.

Selecting the most appropriate programme is less simple. If we were to base our selection on the cause of the switch, the process would be complicated. For example, we may require to give control to:

(a) A lower priority programme - if the current programme cannot be continued.

or (b) A higher priority programme - if a peripheral unit has become available for this programme.

or (c) The same programme again - if a peripheral unit has become available, but for a lower priority programme.

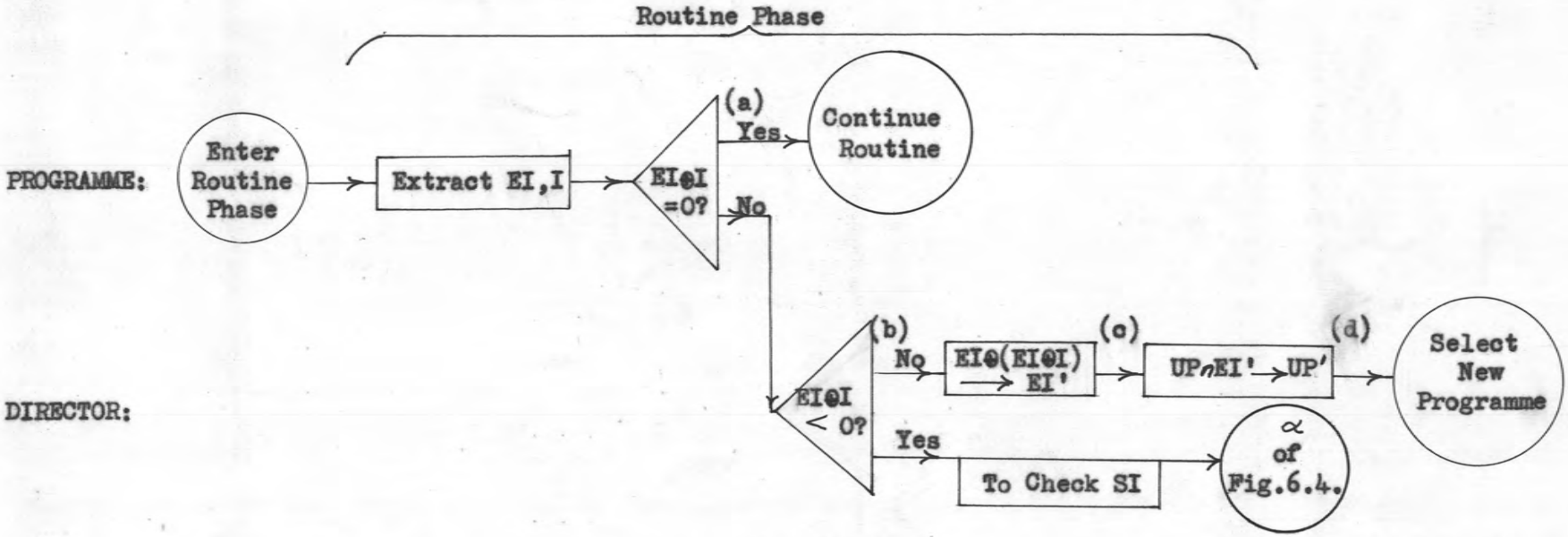


Figure 6.3.: Diagram of the procedure involving the detection of a changed Primary Indicator. (Example 6.2.)

Notes on Figure 6.3.:- (a) Every non-zero digit of EI@I (other than the first represents

either (i) A unit which has completed a transfer,
or (rarely) (ii) A unit which has become unavailable through some cause external to the computer.

(b) This checks for a possible Secondary Indicator change.

(c) EI has been amended to allow for the changed conditions.

(d) UP now records only units which are still unavailable.

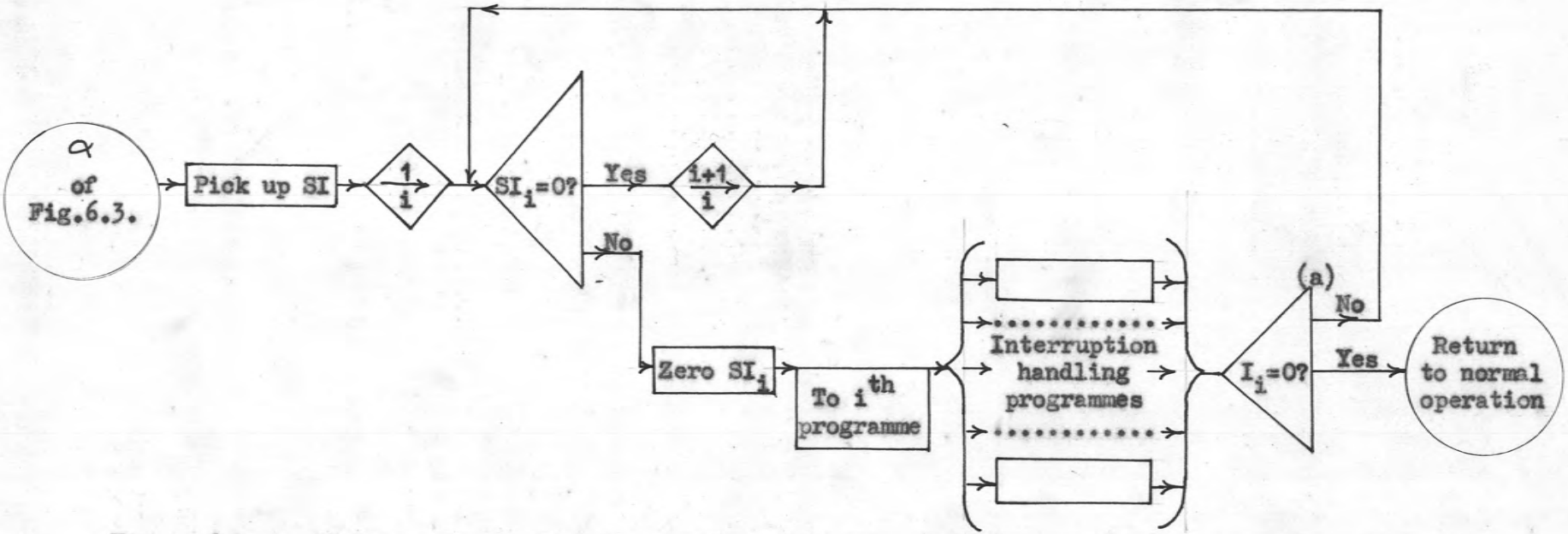


Figure 6.4.:- Diagram of procedure for dealing with interruptions caused by a change in a Secondary Indicator.

Notes on Figure 6.4.: (a) This checks that all other SI_i are zero. If not, the inspection of further SI_i should be continued from the point where this non-zero one was found.

Note also that the order in which the Secondary Indicators are allocated establishes the priority with which the interruptions are dealt.

or (d) An entirely new programme - as a result of some external intervention.

or (e) No programme - if with condition (a) above, there is no lower priority programme.

To simplify matters, programme selection on CIRRUS will always follow the same procedure. The programmes will be examined in descending order of their priorities, and the first programme which can be followed will be selected. All sequence counters will be held in priority order in the "sequence counter ladder" - a set of successive main store words. Programme switches will be confined largely to interchanging the two or three programmes highest in priority, so that this procedure should be quite efficient.

This raises the question of what is to be the criterion to determine whether or not a programme can be followed. In general, a programme can operate unless its next order requires a peripheral unit which is unavailable. The record of "Unavailable Peripheral Units", UP, is added to only when a unit is found to be unavailable in the execute phase of a transfer order. This order is still to be obeyed in this programme. UP is readjusted as units become available, so the decision whether a programme can be followed is based on whether UP contains any units which are used by the programme.

Definition: The Sequence Counter Word.

A single word, of which there will be one for each programme. The set of sequence counter words is arranged in a priority order on the sequence counter ladder.

Each word will contain:

Upper Half	(Bit 1	Programme Lockout Code.
	(Bits 2-18	Relevant Indicators, RI.
Lower Half	(Bits 19-22	Programme Number.
	(Bits 23-36	Sequence Counter, SC.

Definition: The Relevant Indicators, RI.

This is a record of the peripheral units used by the programme. Each non-zero bit corresponds to the indicator of the unit.

We can now say that the j^{th} programme can be selected if

$$RI_j \cap UP = 0,$$

During the routine phase, the source of the address of the current sequence counter will be the E register. Hence, we wish to set E to the address of the first RI_j satisfying the above criterion. The actual programme selection procedure is shown in Figure 6.5. It is assumed that the address of the highest priority sequence counter is n.

The first bit of UP will always be kept as 1. This will make it possible for a programme to be locked out

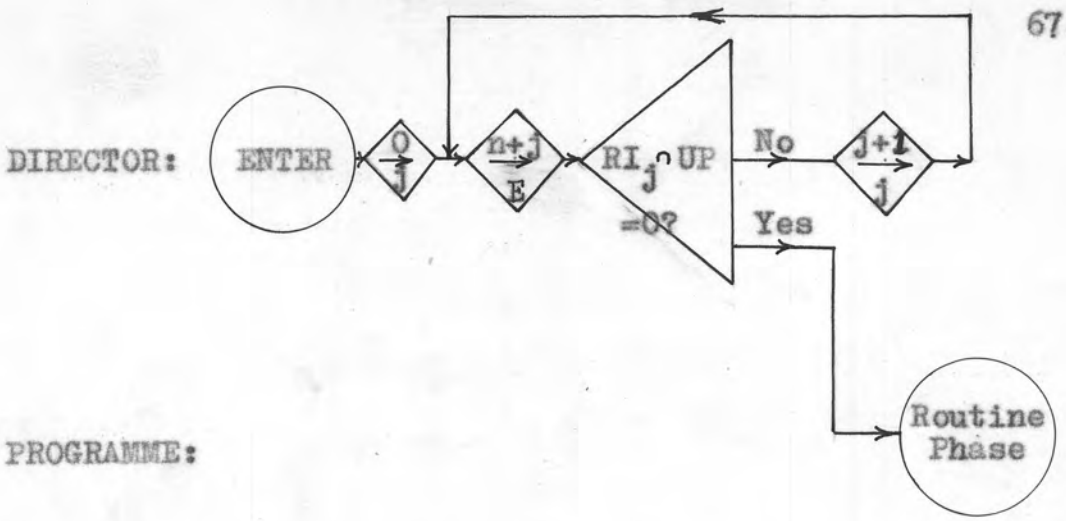


Figure 6.5.:- Diagram of programme selection procedure.

simply by setting the first bit of its sequence counter word equal to 1.

To cater for the occasion when no programme can be operated, the word following the last valid sequence counter will have the form

RI = 0 - i.e. the "programme" can operate.

SC = x - address x (in fixed store) holds the order "jump to x".

Continual execution of this jump order will occur until a change of indicators is detected in the routine phase. This operation is termed "routine cycling".

The programme number incorporated in the sequence counter words can vary from 0 to 15. This number does not refer to the programme's position on the sequence counter ladder. Routines originated by the Director will have numbers greater than 8, while working programmes, of which there could be 7, will be numbered from 1 to 7. With these working programmes, the programme number refers

to the position in which the book-keeping information appertaining to this programme is held by the Director in a "programme catalogue".

Sequence counters for Director routines will always precede those for working programmes. Sequence counters for programmes in either category will always begin by occupying the lowest priority position of their group. To handle the reshuffles which will occasionally be necessary, the Director will have available a micro-ordered programme, which, depending on the point of entry, will:

(a) Put a specified word in the position immediately following the sequence counter of the lowest priority Director programme.

or (b) Put a specified word in the position immediately following the sequence counter of the lowest priority programme.

or (c) Remove the current sequence counter.

After any of these alternatives, control will go to the Programme Selection Routine, as the appropriate programme will now almost certainly not be the one which was last operated. Of the above alternatives, the first is used to bring a Director routine into operation, or to raise the priority of a particular working programme. (All working programmes will begin as a Director routine). The third is used to end a programme.

6.1.3. Duration of a Programme Switch.

Having specified a scheme for programme switching, some estimate can be made of the time which this will take. The relevant period is that from the completion of the last machine order of one programme to the commencement of the first order of the next. Let us consider an example:

Example 6.4.

Operation of a programme m^{th} in priority is halted by an unavailable unit and control is switched to a programme n^{th} in priority ($n > m$), returning to the former programme when the unit becomes available again. A diagram showing the number of main store cycles necessary is given in Figure 6.6.

It can be seen from Figure 6.6. that in the first switch, $4 + n$ main store cycles are used, and, in the second, $3 + m$. Assuming a main store cycle time of 7 μ sec., and adding 30% to cover those operations which cannot be done simultaneously with the main store transfer, approximate times for the two switches are $(36 + 9m)$ μ sec. and $(27 + 9n)$ μ sec. respectively. Probably a majority of programme switches will be made from the top priority programme to the second priority programme and, later, back again. The two switching times in this case would be 54μsec. and 36 μ sec. respectively

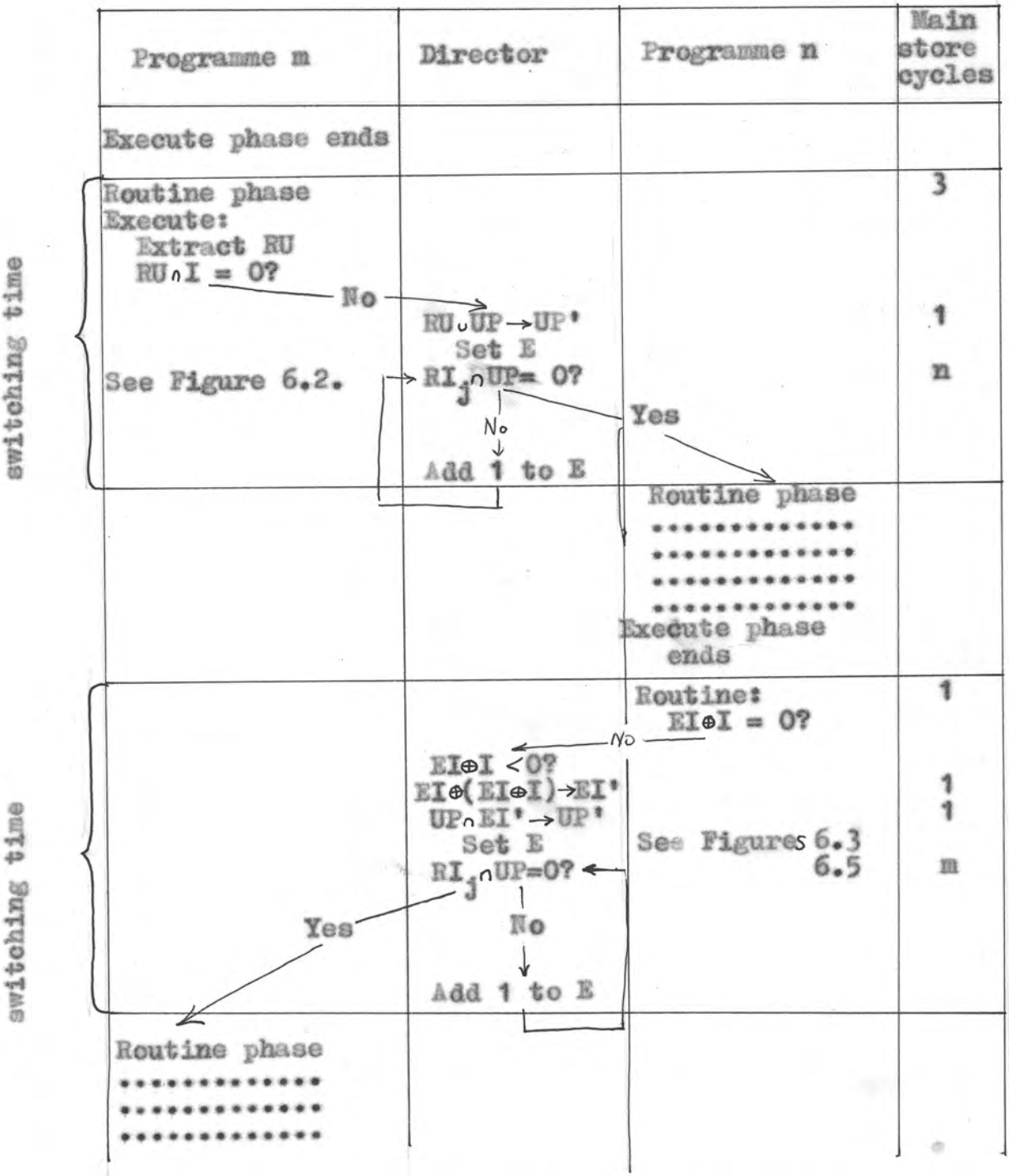


Figure 6.6.:- Diagram showing the number of main store cycles involved in programme switching operations.

6.1.4. The Form of the CIRRUS Director Programme.

In the examples of the preceding sections, the Director operations are programmed with micro-orders rather than machine orders. This is desirable because

(a) Transition from programme to Director is made as the result of a micro-ordered test.

(b) The frequency with which each of these particular operations will be performed makes it necessary for them to be as rapid as possible.

(c) Using machine orders, it would be necessary in many cases to suppress the checking of indicators during each routine phase. Allowing the Director to be diverted to deal with another request for action would, in certain circumstances, considerably complicate matters.

However, with longer Director operations, the micro-code, being limited in scope, would be cumbersome to use. Hence, normal machine orders are used. The change from micro-orders to machine orders is usually made through the programme selection routine, which, as has been shown, sets the appropriate sequence counter address, and then transfers control to the head of the routine phase.

As the Director programme is to be held in the fixed store, some method must be found of coping with what is perhaps the prime problem associated with fixed store use; that is, the supplying of parameters to each fixed pro-

gramme. Multiprogramme operation makes this problem more acute. The possibility of a programme switch during the operation of a fixed programme makes it desirable that its working space should be tied to that particular programme of which the fixed programme is, temporarily, a part. This means that working addresses for fixed routines should also be expressed parametrically.

How this parametric addressing will be achieved has not fully been decided, as it involves considerations not directly connected with multiprogramme operation. However it will probably be brought about by using a variation of the routine phase with orders from fixed store. The fact that an order is to come from fixed store is known because the order address, brought from the store in the first cycle of the routine phase, will be greater than 16,384. In this routine phase there will be provision for a parameter or parameters to be added to the various store addresses if desired.

This procedure will increase the time required to obey orders when parameters are to be added. However the fixed store will become considerably more versatile. In particular, it will mean that parts of the Director can be used "simultaneously" with several programmes; for example, after a little preliminary organisation it will be possible for two or three programmes to be assembled concurrently.

6.2. Operating CIRRUS.

Because of CIRRUS' multiprogramme structure, any instruction to the computer must usually give more information; the operator, if wanting a programme to be read, must tell the Director which reader holds his programme tape, or, if wishing to act on a programme already operating he must identify the programme for the Director.

Certain aspects of the operation of a multiprogramme computer make it desirable that operator communication with the Director should be a two-way affair. For example, if a programme fails to read on a single programme computer, the operator will search for some fault; if it fails to read on CIRRUS, the operator will no doubt assume that his programme cannot be accommodated; however, this may not be the case. Hence, the Director should inform the operator when it decides it cannot read a programme and state the reason.

To achieve these higher degrees of operator-computer communication, the usual console handkeys are inadequate. The CIRRUS consoles, as shown in Figure 6.1., will include input/output writers. A suggested unit for these "writers" is the Siemens Page-Printer, Model 100, modified so that its key-board and typewriter are independent. Each key-board and each typewriter will have its own single character buffer, so that their operation will, to the computer, be identical with that of a paper tape reader and punch.

With the keyboard, the operator will send "instructions" to the Director, and on the typewriter, the Director will issue "directives" to the operator.

6.2.1. Operating Instructions.

The instructions which the operator can give to the Director will form a certain, definite set. As will be seen, the number of instructions which may be provided is limited only by the number of keys on the key-board, and the capacity of the fixed store to hold programmes to handle them.

The following is a list of the projected instructions:

(a) Read instruction.

(b) (i) Read programme.

(ii) Raise priority.

(iii) Suspend programme.

(iv) Resume programme.

(v) Reject programme.

(vi) Obey the order following.

(vii) Post Mortem.

(viii) Display current programmes.

(c) Register programme number.

(d) (i) Print instruction.

(ii) Obey instruction.

Each instruction will have a key on the keyboard, labelled with the instruction. Except for the keys corresponding to the instructions under (a), (c) and

(d) above, each key can, if necessary, also correspond to either an alphabetic or numerical character.

To send an instruction, the operator will press in turn:

(a) The key: "Read instruction".

(b) The instruction key itself, followed by any further information required.

(c) The key: "Obey instruction".

Pressing the key, "Read instruction", a second time before pressing the key, "obey instruction", will cause the cancellation of whatever instruction has been assembled. Use of the "print instruction" key, between step (b) and (c) above, will enable the operator to check his instruction before causing it to be obeyed.

Let us now consider the way in which the Director will read and act upon these instructions. Pressing the key "read instruction", will set a Secondary Indicator. This will activate a micro-ordered programme which will put a sequence counter word on the sequence counter ladder. The form of this word will be :

Sequence counter = x, Relevant Indicator = i,

where x = address of head of a "read instruction" routine.

i = (primary) indicator corresponding to the keyboard.

This "read instruction" routine, which is diagram-

matically shown in Figure 6.7., will read characters sent from the keyboard, and assemble them, character-by-character in a special "instruction store", of which there will be one for each reader. Each character will be checked to see whether it is "print instruction", "obey instruction" or "register programme number".

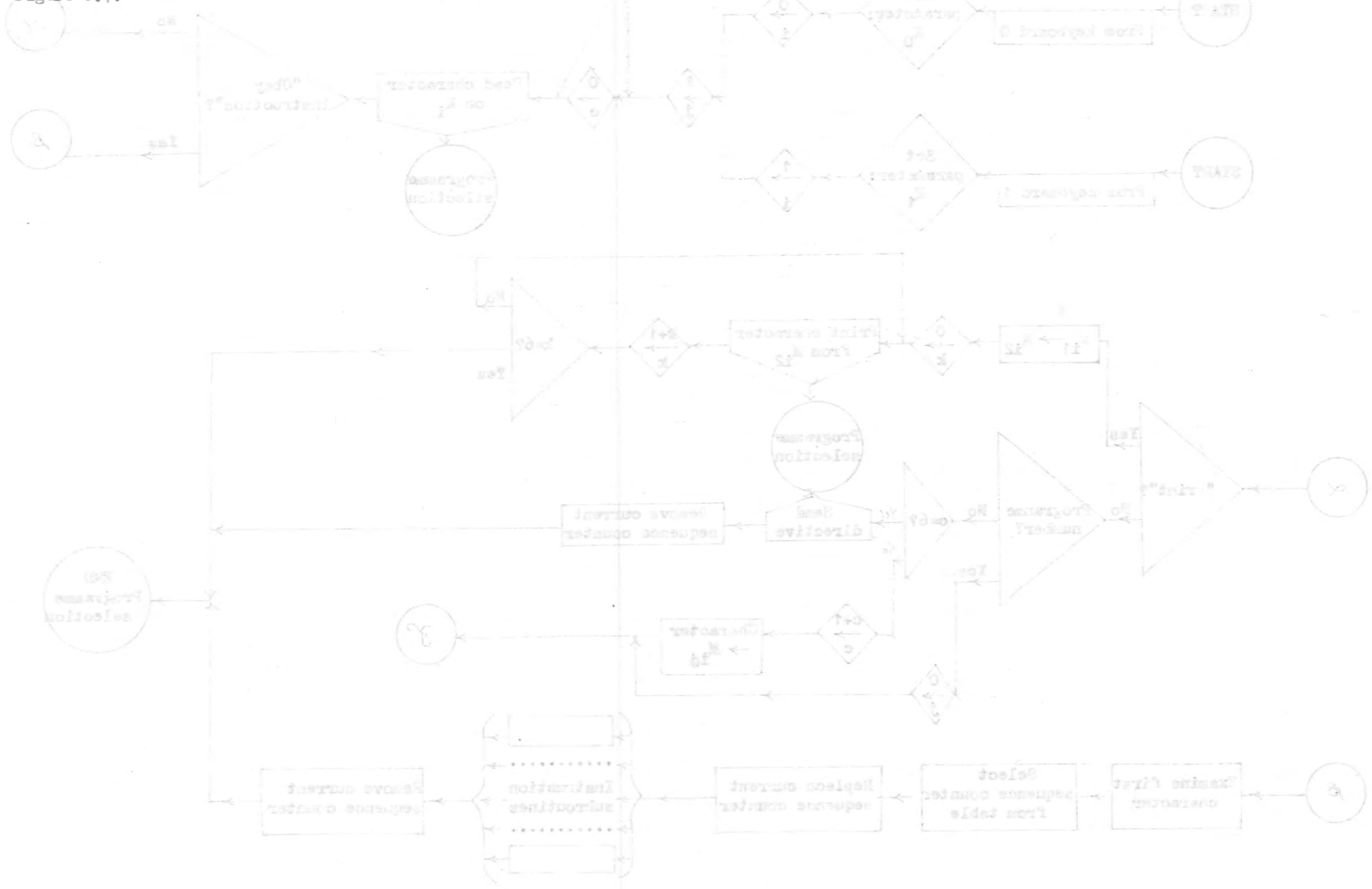
The instruction store for each keyboard will comprise 3 locations in the reserved part of main store. The first location will hold the programme number, to be filled only after "register programme number" has been detected. This means that each programme number can consist of up to six characters, each of six bits.* The virtue of storing the programme number separately is that the operator need specify it only once in any sequence of instructions referring to the same programme.

The second location will hold the instruction, similarly assembled in character-by-character form. Detection of the "print instruction" character will result in the instruction being printed on the typewriter. During this operation, the third instruction store location is used as a working location. Detection of the "obey instruction" character, will cause the first character of the full instruction (the operative character of

---

*This assumes that the character code will have seven bits, one being a parity bit.

Figure 6.7.



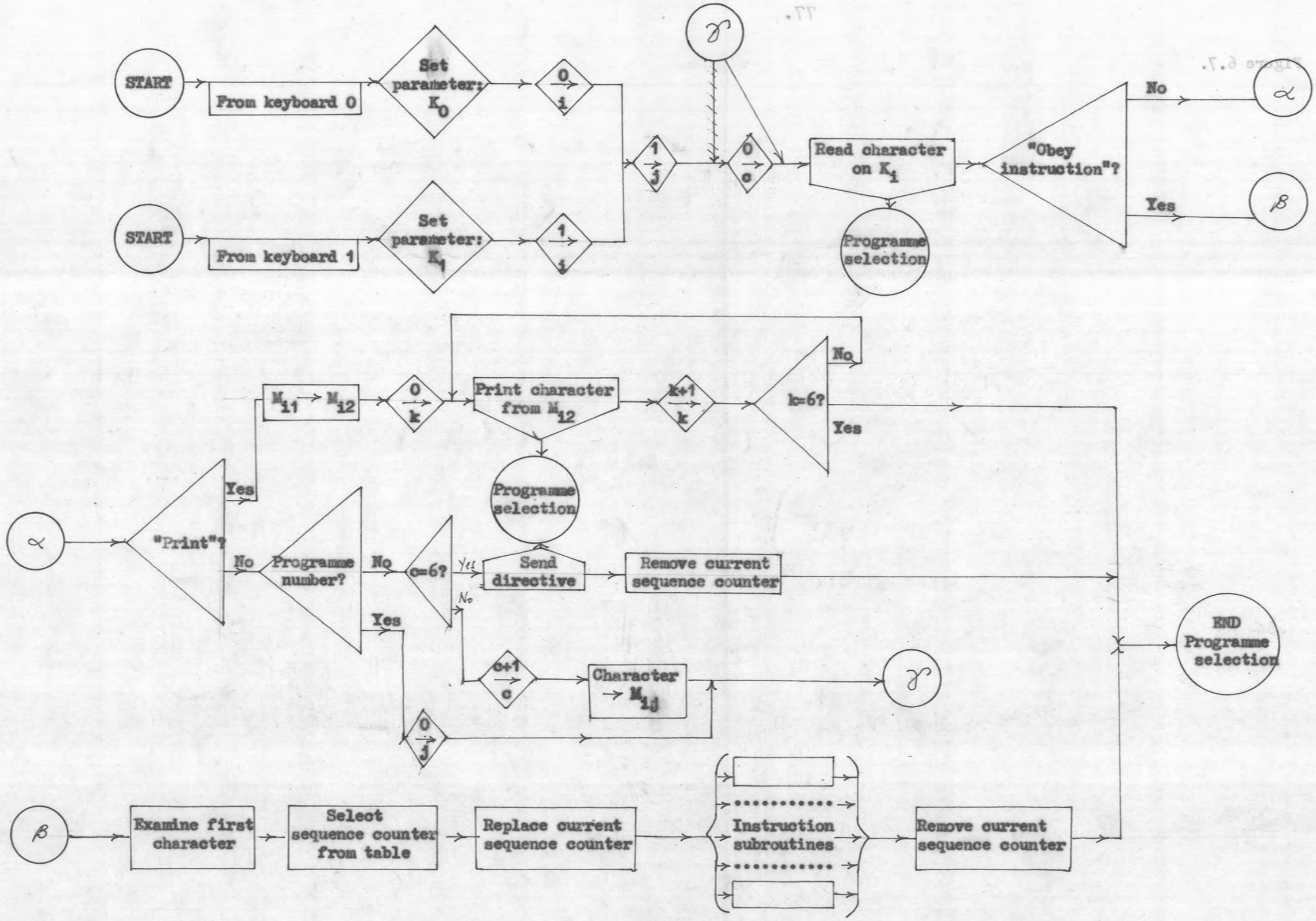


Figure 6.7.:- Sequence of operations in the "Read Instruction" programme.

Notes: The instruction store addresses are M_{ij}, i=0,1; j=0,1,2.

M_{00}: Keyboard 0-Programme No.; M_{10}: Keyboard 1-Programme No.

M_{01}: " Instruction ; M_{11}: " Instruction

M_{02}: " Working Location ; M_{12}: " Working Location

the instruction), being used to select a sequence counter word from another fixed store table. The sequence counter word, which will replace the one used to read the instruction will refer to a Director sub-programme which will carry out the instruction.

Let us now consider what action the Director will take on receiving each of the instructions mentioned under (b) in the set of instructions.

Instruction (i) Read programme.

This instruction must also include the number on which the programme is required to be read. Use of this instruction will lead eventually to the preliminary section of the assembly routine. This section will decide whether a programme can be read, and, if so, will bring the actual assembly programme into operation.

(ii) Raise Priority.

Any programme shall be considered as having commenced when the initial section of the assembly routine has found that the programme can be run, and has allotted various parameters to it. From this stage, during assembly and operation, the programme will normally be of lower priority than any other programme which has preceded it. This is to say that its sequence counter, which will initially be the assembler sequence counter, will occupy the lowest position on the

sequence counter ladder. However, use of this particular instruction will cause the sequence counter to be moved above all sequence counters other than those for Director sub-programmes, and hence take precedence.

(iii) Suspend programme.

This will allow for a single programme to be "locked-out", without interference with operation in general. This effect will probably be achieved by setting the first "Relevant Indicator" bit of the sequence counter word equal to 1, and keeping the first bit of "unavailable Peripheral Units" always equal to 1. The test

$$RI \cap UP = 0?$$

will therefore always fail and the programme will not be selected.

An order, available to the programmer, will also achieve this effect, replacing the "stop" orders of conventional computers, used for example when a new tape is to be loaded in the reader.

(iv) Resume programme.

This instruction will cancel the "suspend programme" instruction by removing the initial non-zero bit in RI. The programme will become operable again but will, of course, not necessarily come immediately into operation.

(v) Reject Programme.

If the operator wishes to discontinue a programme, this instruction will cause the removal of its sequence counter from the sequence counter ladder, and the readjustment of the records of what store space and peripheral equipment are available. (The same routine is used at the conclusion of any programme.)

(vi) Obey the order following.

On conventional computers, hand-keys are generally available by which an experienced programmer can cause the computer to obey any desired order. This is often useful in programme testing. To duplicate this facility on CIRRUS, a special instruction will be needed. This instruction can be incorporated with the assembly routine, so that the standard punching code can be used to specify the order.

(vii) Post-mortem.

Post-morten routines used with CIRRUS will probably be fairly comprehensive, and make use of several instruction keys. The question of post-mortems will be considered later.

(viii) Display current programmes.

This instruction will cause the computer to display, for each programme in the computer, the programme number and the first main store and first register store locations used and conclude with a summary

of the total numbers of locations used.

Of all the instructions mentioned above, some (notably a, b(i), (iii), (iv), (v), c, d(ii).) may be regarded as essential before the computer can be operated. Others, (b(ii), (viii), d(i).), are highly desirable, and not difficult to provide. The remaining two, "obey the following order" and "post-mortem", may prove fairly complex and providing them will probably be delayed. The system for handling operating instructions is such that it will be simple for these or any other instructions to be added at a later date without confusion.

6.2.2. Directives.

In response to certain specific situations, the Director will issue statements to the operator; these statements are to be called "directives", and they will be printed on the typewriters mentioned earlier. The need for a directive will usually appear as the result of a test or check of some kind in the course of a programme, or in response to an operator's request for information.

The directive may be either a fixed message, a display of values of certain variables, or, perhaps, a mixture of both. This can be seen in the examples of proposed directives given below. Where appropriate a directive will be preceded by the programme number.

Depending on the circumstances, it may be made on all typewriters, or on only one.

Unlike other peripheral units, the input/output writer may be used by any programme in the computer. As typewriter operation will time-share the computer, it is possible that, while one directive is being printed, another programme may require a directive. To prevent clashing, an "indicator" (probably a single bit of a specific main store word) must be set at the beginning and reset at the end of a directive. This "indicator" must be queried before printing of a directive is begun.

Examples of occasions requiring directives:

(a) During assembly. If there are already 7 programmes, or if there is not sufficient store space or peripheral equipment, the operator will be told. Details of the allocation of peripheral units will form another directive.

(b) At the end of a programme. The notification ".....ends" will be made. In addition, information on programmes still occupying the computer, together with the store locations used will be supplied. This latter information will also be given when an operator requests it with operating instruction (b)(viii) as described in the previous section.

(c) During post-mortems. Information will be made available according to the request of the operator.

(d) During computation. Attempting to use a zero divisor, take the square root of a negative number, or the overflow of a register may also result in directives. It is likely in practice that with each of these situations the programmer will be able to specify whether or not he wishes notification to be given to the operator.

6.2.3. Further Use of the Input/output Writer.

It will be possible to use the input-output writer for general input and output of data. The keyboard is particularly suited for the input of limited quantities of data. This will not interfere with the keyboard's prime function as a source of operating instructions. As the "read instruction" programme will take priority over the other programmes, it will be possible to call it into operation for an instruction even if the operator is half-way through the input of a number.

The typewriter will be less useful. A directive may occur at any time, regardless of whether or not the typewriter is already in use. It would therefore be necessary for directives to be intermingled with data. The directives could be distinguishable from data if the computer were able to control the ribbon colour, but their presence would make the data display messy. Hence the typewriter will be useful only for relatively un-

important data displays, such as those from post-mortem or maintenance test routines.

6.3. The Assembly and Operation of a Set of Programmes.

Let us now consider the operation of a set of programmes, beginning with the computer as it is when first switched on. When the computer is ready to operate control will be given to the Director. The Director will set a few basic values, such as UP, which as stated earlier must begin as 10....0, and then settle down into the routine cycling operation. The operator, or operators may then break into this routine cycling to send the instruction "read programme".

The programmer cannot specify where his programme is to be stored. Each programme when read, will be given a "base address" in each of the main and register stores. For the first programme, the base addresses will be the first addresses following the sections of store reserved as working space for the Director. For subsequent programmes, the base addresses will be the first addresses available after allowing for all the requirements of the preceding programme.

When a stage is reached where the next programme to be read cannot be accomodated, either because of shortage of space in one or both of the stores, or because insufficient peripheral equipment is available, this programme cannot be read until at least one of the cur-

rent programmes is concluded. The programmes are unlikely to end in a convenient order, so we are faced with the likelihood of having vacant store sections which may be difficult to use.

For future operation from this stage, three possible courses may be considered.

(a) We can wait until all programmes are completed, and then recommence our store loading cycle.

(b) We can attempt to make partial use of these vacant sections by fitting in any shorter programmes which may be available.

(c) Beginning with the programme following the first vacant section of store, we can move this and subsequent programmes up in the store, thus consolidating all empty store space at the end of each store.

The first alternative is, of course, the simplest, but is an inefficient arrangement, particularly in the case where there is one programme of considerably longer duration than the others. Such a programme would necessarily run for most of its time on its own in the computer. The second alternative would result in more efficient use of the store, but would be fairly difficult to apply in practice.

Choice of either of these two alternatives would make it desirable for some consideration to be given to what would be the most suitable programmes to be run

together. This is contrary to our general policy whereby each operator should be able to work without considering what any other operator is doing.

Adopting the third alternative would mean that the store is used in the most efficient way possible. However, carrying out this "shifting up" procedure which is, to some extent, a reassembly of the programmes in the store, presents certain difficulties. These will be discussed after the assembly of programmes has been described. It is felt at this stage, that the benefits of this process of re-assembly, or, as we shall call it, "store reorganisation", will over-ride its disadvantages.

Let us assume, then, that store reorganisation will be done if the operator wishes to read a programme, and:

(a) There are insufficient consecutive addresses at the end of either store.

but (b) There would be enough addresses if all vacant addresses were consecutive.

and (c) There is no other reason to prevent the programme being read.

6.3.2. Programme Assembly.

Detection by the Director of the "read programme" instruction from an operator will result in the sequence counters of the instruction reading programme being replaced by a sequence counter referring to a rou-

tine for assembling programmes. Prior to any actual programme assembly, there will be a preliminary section of the assembly routine. This is illustrated diagrammatically in Figure 6.9. First, a parameter will be set for the subsequent assembly, showing which reader has been requested as the programme reader. After this, a decision will be made on whether the computer can accommodate the programme. For this to be possible:

(a) There must be less than 7 programmes already in the computer.

(b) There must be adequate peripheral equipment.

(c) There must be sufficient store space.

To decide these last two questions, the assembly routine must know the programme's requirements. This information will be available as a heading on the programme tape. The heading will comprise:

(a) Programme reference. PR.

(b) Programme "status" code.

(c) Store space required in:

(i) Register store, r.

(ii) Main store (programme), n.

(iii) Main store (data), m.

(d) Peripheral equipment required.

Preparation of this heading would be done by the compiler routine for a programme being automatically compiled. Probably a special programme will be used for preparing the heading for an individually written programme. In this latter case, while sections of the programme are being tested, an estimate of store space may be used which need not be accurate but must be in excess. The "status" code referred to above, will tell the Director whether or not the programme has been proven; if still under test, the Director will arrange that all its operand addresses are tested before any order is obeyed to ensure that they do not interfere with another programme.

If in the prelimary section of the assembly routine the tape heading is read, and it is found that the programme can be accommodated, three more parameters will be set showing the base addresses for main store (programme), main store (data) and register store. After this the sequence counter for the assembly routine will be placed at the bottom of the sequence counter ladder and reading of the programme tape and assembly of the programme can commence. From this stage the assembly routine will be regarded by the Director as one of the working programmes; it will be able to use, for its own working locations, the data addresses allotted to

the programme which it is assembling. Transition from assembly to operation of the programme will be achieved simply by changing the value held in the sequence counter.

After the preliminary section has been passed, the assembly routine may therefore be regarded virtually as a subroutine of the programme which it is assembling. Providing certain quantities are expressed parametrically in the routine, assembly of more than one programme at once will be possible. However, the preliminary section must not attempt to operate on more than one programme at once. How this will be prevented can be seen from the notes on "waiting programmes", given with Figure 6.9.

6.3.3. Store Reorganisation.

Before going on to discuss the difficulties associated with the process of store reorganisation, let us consider certain aspects of the modification of orders by addition of base addresses during programme assembly. The assembly routine must not add a base address where:

(a) The "address" does not refer to a store address. Orders where this is so are:

(i) Input/output orders, where Y identifies the unit.

(ii) "Shift" orders where X is the number of shifts.

(iii) "Set constant" orders where X is the constant.

(b) The order is a "pseudo-order" or programme constant.

(c) The address refers to the fixed store.

(d) The address is zero.

To distinguish these cases, tags or parameters may be punched on the programme tape, or, alternatively, the assembly programme may itself make the distinction.

When a programme is to be shifted bodily during store reorganisation, its base addresses are altered, and its operand addresses must be adjusted accordingly. The exceptional cases mentioned above again must be distinguished. Programme tape tags or parameters are no longer available, so the identification must be made by the programme. This can be done without a great deal of difficulty. To give uniformity the identification procedures adopted for assembly and reorganisation should be the same. Allowing the assembly and reorganisation programmes to decide, by examination of the function and address digits of an order whether or not base addresses should be added means that in writing programmes certain restrictions must be observed:

(a) Numbering of both main and register store addresses in the programme must begin at 1, not 0, to distinguish cases where the address is genuinely zero.* However, due allowance may be made for the case where the address is given as zero, but there is B-line modification.

(b) The programme must not modify the function digits of an order if the two orders are to have different treatment as far as addition of base orders is concerned.

(c) The programme must not, in adding to an address of an order, change it from a main store to a fixed store address.

In reorganisation, there is the additional complication that all data values must be shifted without amendment. This means that there must be separation of programme and data in the store.(This is the reason why requirements in both programme space and working space (n,m) are included in each programme tape heading.) Where the main store base address of the programme is N, data addresses will begin at N + n.

The assembly routine must store programme constants in

---

* This means that the addresses added to each order will in fact each be less by one than the actual base addresses.

the data section, starting most conveniently from N+n. Constants will be punched numerically and no other identification to the assembler should be needed. Within sub-routines, the programmer will make reference to constants relative to the routine only. For instance, in referring to the third constant of the routine, he may write "C3". As the assembler will know the address in which the first constant of the routine is to be stored, it can substitute the relevant address in place of "C3".

It will be assumed when moving values in the register store that all values are data values. Reorganisation must therefore be restricted to a point where no orders are held in the register store. To ensure this, it will probably be necessary for the programmer to include at intervals throughout the programme an order showing that no orders are in the register store. As reorganisation would be necessary only once for every second or third programme read into the computer, it does not matter if the delay before it occurs is a millisecond or a minute. These special orders may therefore be put at any points which happen to be convenient.

The restrictions mentioned earlier and the necessity to add these special orders are, in themselves, trivial impositions on the programmer. The chief disadvantage lies in the possibility that a programmer may forget to

observe restrictions, Programme compiler routines would, of course, automatically take them into account. In the author's opinion the considerably improved efficiency of store utilisation, more than compensates for the loss of programming flexibility.

6.3.4. Book-keeping Information.

To enable the Director to make its decisions, it must store a considerable amount of "book-keeping" information. Information which is related to individual programmes is held in an array of addresses termed the "programme catalogue" (Figure 6.8.). It follows from the way in which this catalogue is compiled that programmes will occupy the same relative positions in it that they do in the store. Any vacant section of the store will correspond to a "blank" space in the programme catalogue. (A "blank" space will probably be signified by the programme reference number being zero.) The programme catalogue will also store certain other quantities relating to specific programmes.

The remaining space in the 8 x 8 array shown in Figure 6.8., is taken by other Director quantities. Quantities ro, no, mo, PRo are the values read from the heading of the programme tape during the preliminary section of the assembly routine, After it has been ascertained that the programme can be read, it is given a

Figure 6.8.

GENERAL VALUES	PROGRAMME CATALOGUE							Upper/ Lower
	PROG.1	PROG.2	PROG.3	PROG.4	PROG.5	PROG.6	PROG.7	
0: r_0	1: r_1	2: r_2	3: r_3	4: r_4	5: r_5	6: r_6	7: r_7	Upper
0: n_0	1: n_1	2: n_2	3: n_3	4: n_4	5: n_5	6: n_6	7: n_7	Lower
8: m_0	9: m_1	10: m_2	..	..	..	..	..	Upper
8: -	9: M_1	10: M_2	..	..	..	..	..	Lower
16: PR_0	17: R_1	18: R_2	..	..	..	..	..	Upper
16:	17: N_1	18: N_2	..	..	..	..	..	Lower
24:	25: PR_1	26: PR_2	..	..	..	..	..	Upper
24:	25:	26:	..	..	..	..	..	Lower
32:	33: L_{11}	34: L_{21}	..	..	..	..	..	Upper
32:	33: L_{12}	34: L_{22}	..	..	..	..	..	Lower
40:								
48:								
56:							63:	

Figure 6.8.:- Storage of book-keeping information, including the programme catalogue.

Notes on Figure 6.8.:-

(i) A total of 64 addresses is used, the relative addresses being shown in this figure.

(ii) The suffix i refers to the i^{th} programme, and:

- r_i is the number of register store addresses used
- n_i is the number of main store addresses used for orders
- m_i is the number of main store addresses used for data
- M_i is the first main store address used for data
- R_i is the first register store address used
- N_i is the first main store address used
- PR_i is the programme reference (being up to six 6-bit characters)
- r_0, n_0, m_0, PR_0 are the quantities read from the heading of the programme tape
- L_{i1}, L_{i2} are subroutine links

(iii) Addresses 8(lower),24,32,40,48,56 will be used for other book-keeping quantities.

(iv) 40+i,48+i,56+i will be used for other quantities relating to specific programmes.

"programme number" (see "Sequence Counter Word" in Section 6.1), which corresponds to the programme catalogue position following that of the programme which is last in the store. On deciding what the base addresses for the programme are to be, the assembly routine will include these in the correct positions of the programme catalogue and also move across the quantities r_0, etc, to their correct position.

As examples of book-keeping quantities which are required by the Director, let us define:

(a) First Available Addresses.

A single word showing the first addresses which can be used by the next programme, It will be referred to in the preliminary section of the assembly routine to check whether there are sufficient consecutive addresses. It will be amended when actual assembly begins, and also after a store reorganisation.

(b) Total Store Available.

A single word, showing the total number of addresses available in each store. It will be referred to if it is found that there is insufficient store space at the end of either store for a programme, to discover whether store reorganisation should be carried out. It will be amended when a programme is read, or when a programme ends.

(c) Peripheral Equipment Available.

An 18-bit half-word, with each non-zero bit corresponding to the indicator of a peripheral unit which is available for allocation to an incoming programme. It will be referred to and amended in the preliminary section of the assembly routine and also amended at the end of any programme. Note that the record of the units used in any particular programme is held in the sequence counter word - the "relevant indicators".

6.3.4. The Allocation of Peripheral Units.

In the preliminary section of the assembly routine, the attempt to allocate peripheral units to a programme will be made prior to the attempt to allocate store space. The reason for this is that we do not wish to reorganise the store unless it is certain that, after it has been done, the programme can be accommodated. Otherwise, we may be re-assembling programmes which, in fact, must be completed before the incoming programme can be read.

In programming, it will be assumed that units numbered from zero are to be used. For instance, if the programmer requires two readers he will call them readers 0 and 1. However, the assembly routine will allocate those readers most appropriate to the circumstances, and must display an "allocation list", showing what has been

allocated, against what has been specified in the programme.

e.g. in this case, it might state:

Reader 2 / Reader 0.

Reader 1 / Reader 1.

Before considering the procedure for allocation of peripheral units, three important points must be considered:

(a) The allocation must be consistent with the physical lay-out of peripheral equipment. It is obviously foolish to allot to a programme requiring one reader and one punch, say, units at opposite extremes of the operating consoles.

(b) The allocation should be the same for any given circumstances. This will be particularly important if magnetic tape is used, for the operator should be able to predict which tape units should be loaded with tape.

(c) The allocation procedure must be such that changes in the composition or lay-out of peripheral equipment can be allowed for with a minimum of confusion.

These requirements are mutually contradictory. To satisfy (a) and (b) we must assume a fixed lay-out, while to satisfy (c), we must be prepared for an occasional change in this fixed lay-out. One simple compromise

solution can be obtained by including in the tape heading, not merely the type of units which are required, but also what units are suitable under the different sets of circumstances. The most suitable allocation is dependent on the position which the operator occupies to operate the computer. This is known to the computer by the key-board from which his "read programme" instruction is sent, and the reader on which programme reading is requested.

With the lay-out of Figure 6.1. we have four possible starting points, viz:

(a) Instruction from key-board 0, programme to be read on reader 0.

(b) Instruction from key-board 0, programme to be read on reader 2.

(c) Instruction from key-board 1, programme to be read on reader 1.

(d) Instruction from key-board 1, programme to be read on reader 2.

Table 6.3. shows what allocations may be made under each of these four initial conditions and assuming the console lay-out of Figure 6.1.

Example 6.6.

A programme requires:

2 readers

1 punch

2 magnetic tape units

Table 6.3.

TABLE 6.3.

PERMISSIBLE ALLOCATIONS OF PERIPHERAL EQUIPMENT

Instruction Source	Programme Reader	Paper Tape Readers		Keyboards		Paper Tape Punches		Typewriters		Magnetic Tape Units		Line Printer	
		No. Required	Allotted	No. Required	Allotted	No. Required	Allotted	No. Required	Allotted	No. Required	Allotted	No. Required	Allotted
K_0	R_0	1	R_0	1	K_0	1	P_0	1	T_0	1	Order of preference: M_0,M_1,M_2,M_3	1	LP
		2	R_0R_2	2	K_0K_1		P_2	2	T_0T_1				
			R_0R_1			2	P_0P_2			1	Any one		
		3	$R_0R_2R_1$				P_0P_1			2	Any two		
						3	$P_0P_2P_1$			3	Any three		
										4	M_0,M_1,M_2,M_3		
	R_2	1	R_2	1	K_0	1	P_2	1	T_0		Order of preference : M_0,M_1,M_2,M_3	1	LP
		2	R_2R_0	2	K_0K_1		P_0	2	T_0T_1				
			R_2R_1			2	P_2P_0			1	Any one		
		3	$R_2R_0R_1$				P_2P_1			2	Any two		
						3	$P_1P_2P_0$			3	Any three		
										4	M_0,M_1,M_2,M_3		
K_1	R_1	1	R_1	1	K_1	1	P_1	1	T_1		Order of preference: M_3,M_2,M_1,M_0	1	LP
		2	R_1R_2	2	K_1K_0		P_2	2	T_1T_0				
			R_1R_0			2	P_1P_2			1	Any one		
		3	$R_1R_2R_0$				P_1P_0			2	Any two		
						3	$P_1P_2P_0$			3	Any three		
										4	M_3,M_2,M_1,M_0		
	R_2	1	R_2	1	K_1	1	P_2	1	T_1		Order of preference: M_3,M_2,M_1,M_0	1	LP
		2	R_2R_1	2	K_1K_0		P_1	2	T_1T_0				
			R_2R_0			2	P_2P_1			1	Any one		
		3	$R_2R_1R_0$				P_2P_0			2	Any two		
						3	$P_2P_1P_0$			3	Any three		
										4	M_3,M_2,M_1,M_0		
Units as numbered in programme:			$R_0R_1R_2$		K_0K_1		$P_0P_1P_2$		T_0T_1		M_0 M_1 M_2 M_3		LP

The operator from key-board 0 requests that it be read on reader 2. The assembler will call this initial condition 0/2.

With the lay-out of Figure 6.1., the tape heading has sections referring to four initial conditions: 0/0, 0/2, 1/1 and 1/2. The assembly routine will bypass the first of these, and begin with the section headed with 0/2. This section will comprise (in some coded form):

Initial Condition 0/2:

$$Ro \;/\; R_2$$

$$R_1 \;/\; R_o, R_1$$

$$Po \;/\; P_2, P_o$$

$$Mo \;/\; M_o, M_1, M_2$$

$$M_1 \;/\; M_1, M_2, M_3$$

Where alternative units are shown, the first available one will be selected in each case. If none of the suggested units are available, the operator will be informed. Unless the operator sends the instruction "reject programme", this programme will remain, with its peripheral units partially allocated, until a return to this preliminary section of the assembly routine is made automatically on the conclusion of a programme.

This allocation procedure satisfies our three requirements reasonably well. If additions or amendments are made to the set of peripheral units, headings might

need to be reproduced, but no change to the fixed programme in the store should be necessary.

6.3.5. The End of a Programme.

At the end of any programme, the programmer will include a jump order giving control to a Director sub-programme. This sequence will also be entered if the operator sends an instruction, "reject programme.....". In this sequence the Director will:

(a) Adjust the "Total Store Available".
"Peripheral Equipment Available"
and, possibly, "First Available Addresses".

(b) Clear the "Programme Reference" location of the programme catalogue to show that this programme has concluded.

(c) Send a directive to the operator:
".....ends"

(d) Give the operator information on the contents of the computer (as described for instruction b(viii) of Section 6.2.)

(e) Check whether another programme is waiting to be read. If so, a jump will be made to the preliminary section of the assembly. If not, the current sequence counter will be removed. Figure 6.9. shows how the "End of programme" sequence merges into the preliminary section of the assembly.

Notes on Figure 6.9.:

"Waiting Programmes."

If a programme is waiting to be read, this fact is known because values are held in locations 0, 8, 16 of the array of book-keeping information. As only one programme may have values in these locations, the test in Step (a), which involves examining location 0, prevents any further programme having access to this preliminary section until Step (b), when the earlier programme has been completely set up for assembly. If a request is made to read another programme, then a note is made (Step (c)) of the reader which holds the programme tape for future reference (Step (d)).

The first waiting programme may be held up because peripheral equipment was not fully allotted in Step (e), or it may have peripheral units allotted and then be held up because of insufficient store space (Step (f)). Which of these was the case must be noted, so that re-entry may be made at the appropriate point (Step (g)).

Figure 6.9.

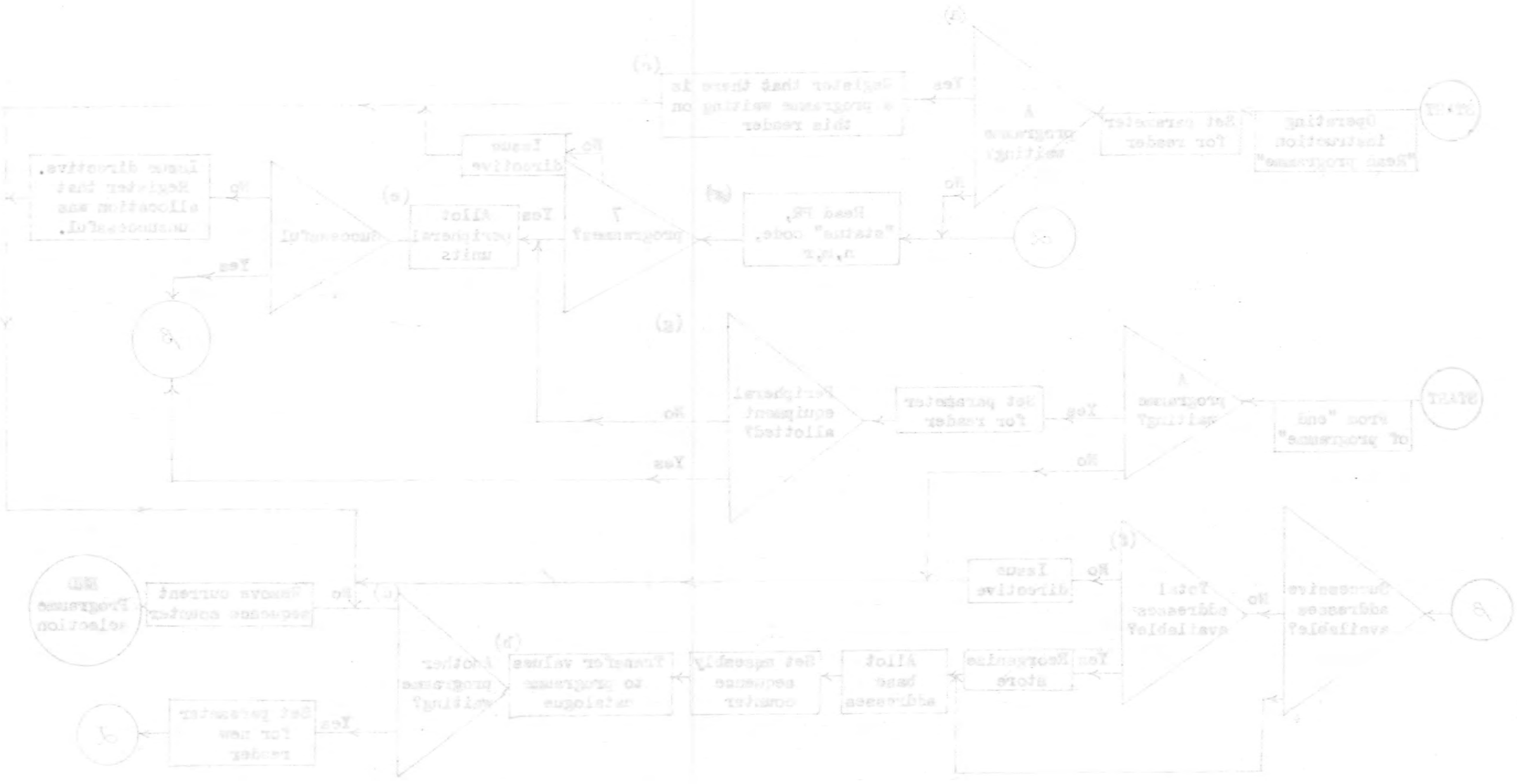


Figure 6.9.:- Diagram showing the steps included in the preliminary section of the assembly routine.

6.4. Extension of the Multiprogramme System to Allow for the Use of Magnetic Tape.

Although it is unlikely that magnetic tape units will be included with CIRRUS initially, magnetic tape is such an important medium and its use presents such distinctive problems in time-sharing, that a method must be suggested by which it can conveniently be incorporated in the proposed system. The details of a magnetic tape handling system will depend heavily on the character rate of the units used, and the degree to which simultaneous operation is desired. For discussion, let us suppose there will be four units, each with a 6-bit character rate of 15 Kc/s* and that simultaneous operation of all four is wanted.

Each unit will have two 18-bit buffers into one of which characters will first be assembled after reading and from the other characters will be taken during writing. In addition there will be secondary buffering, obtained by extending the register store by 68 words. This extension will be simple and inexpensive as it only involves the enlarging of an existing store.

Let us call this extension of the register store the

---

* A unit with this character rate is the widely used Ampex FR400.

"principal" buffer store B. Bo to B_{63} will be actual buffer locations, B_{64} - B_{67} will hold the relevant main store addresses from which, or to which, data is being transferred. The distribution of these addresses B_o to B_{67} is shown in Table 6.4.

TABLE 6.4.

DISTRIBUTION OF BUFFER ADDRESSES B_o - B_{67}.

B_o to B_7:	Input buffer for M_o.
B_8 to B_{15}:	Output buffer for M_o.
B_{16} to B_{23}:	Input buffer for M_1.
B_{24} to B_{31}:	Output buffer for M_1.
B_{32} to B_{39}:	Input buffer for M_2.
B_{40} to B_{47}:	Output buffer for M_2.
B_{48} to B_{55}:	Input buffer for M_3.
B_{56} to B_{63}:	Output buffer for M_3.
B_{64}(upper):	Reading address; (lower): Writing address for M_o.
B_{65}(upper):	Reading address; (lower): " " " M_1.
B_{66}(upper):	Reading address; (lower): " " " M_2.
B_{67}(upper):	Reading address; (lower): " " " M_3.

The timing cycle of the computer will be amended so that,

with a maximum delay of 7μ sec., the contents of a filled half-word input buffer will be transferred to the main buffer store. This transfer will require one store cycle for every third character, or about 4% of the total computer time. The main buffer locations will be filled from 0 to 7, and the filling cycle will restart from 0. During writing, a similar procedure will be followed in reverse. The circuitry used in filling and emptying these main buffers must be independent of anything used by a computation. A diagram showing the hardware associated with a single tape unit is shown in Figure 6.10.

During the execute phase of an order "read from unit M_o to main store beginning at address r_o", the computer will:

(a) Check that the unit is available for the transfer.

(b) Add the unit's indicator bit to UP. This will lock-out the programme until the transfer is completed.*

(c) Amend EI to show that it is expected that the unit will be unavailable.

---

* This is merely a safety device to prevent reference being made to the section of the store involved in the transfer. In practice it may be preferable not to "lock-out" the programme entirely.

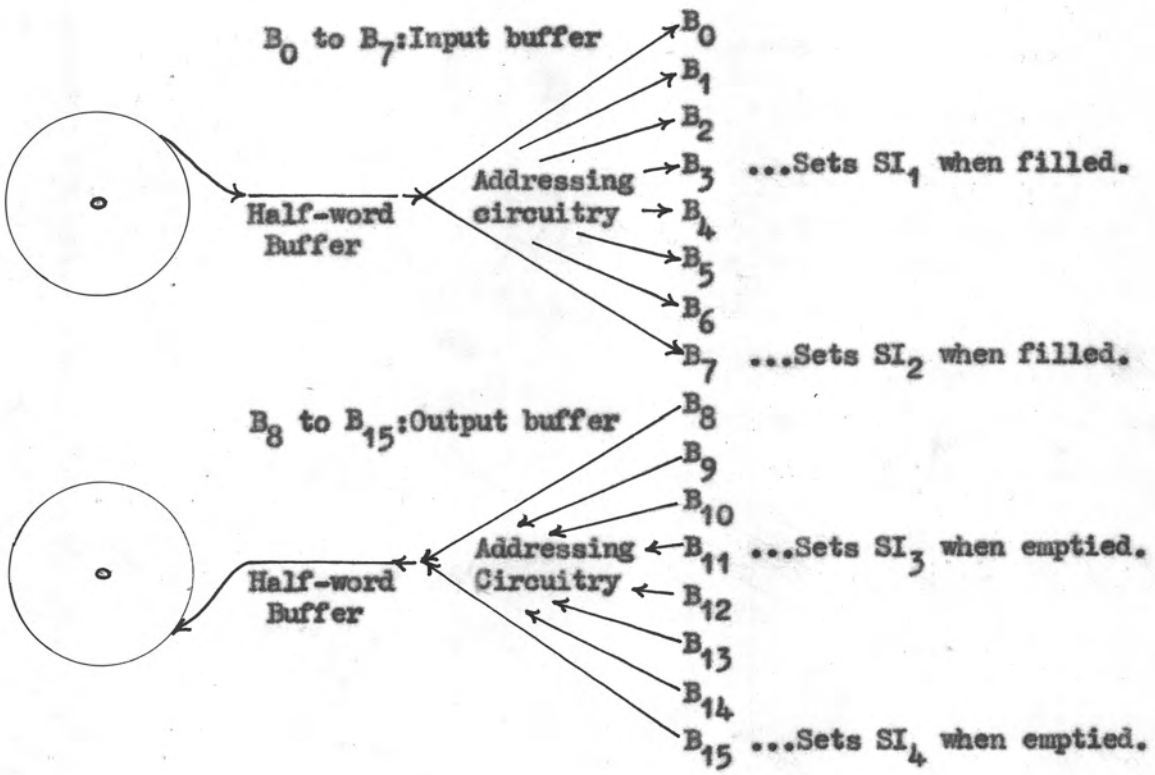


Figure 6.10.: Diagram showing the buffering associated with magnetic tape unit M_0.

(d) Set the address of the first main store location to which data is to be read in the upper half of B_{64} (See Table 6.4.)

(e) Set the tape in motion.

The filling of locations B_3, B_7, or the emptying of locations B_{11}, B_{15} will each set separate Secondary Indicators (See Table 6.2.). Detection of a changed Secondary Indicator will, as described in Section 6.1., result in a programme corresponding to the indicator being brought into operation. The programme corresponding to SI_1 will, during reading, bring down the address r (initially r_o) from B_{64} (upper), increase it to r + 4 and return it, and then transfer the contents of B_0, B_1, B_2, B_3, to main store addresses r, r + 1, r + 2, r + 3. The programme corresponding to SI_3 will increment the quantity (w) in B_{64} (lower) by 4, and use the unincremented value, transferring the contents of main store locations w, w + 1, w + 2, w + 3 to B_8, B_9, B_{10}, B_{11}.

If a double read/write head is used (Figure 6.11), during the writing of a block, data will be read back into the input buffer while writing proceeds. If the unit is writing, instead of transferring the contents of B_0, B_1, B_2, B_3 to the main store when SI_1 is set, these values must be compared against the contents of r, r + 1, r + 2, r + 3. This is why a special read/write indica-

tor is needed to show whether the unit is reading or writing.

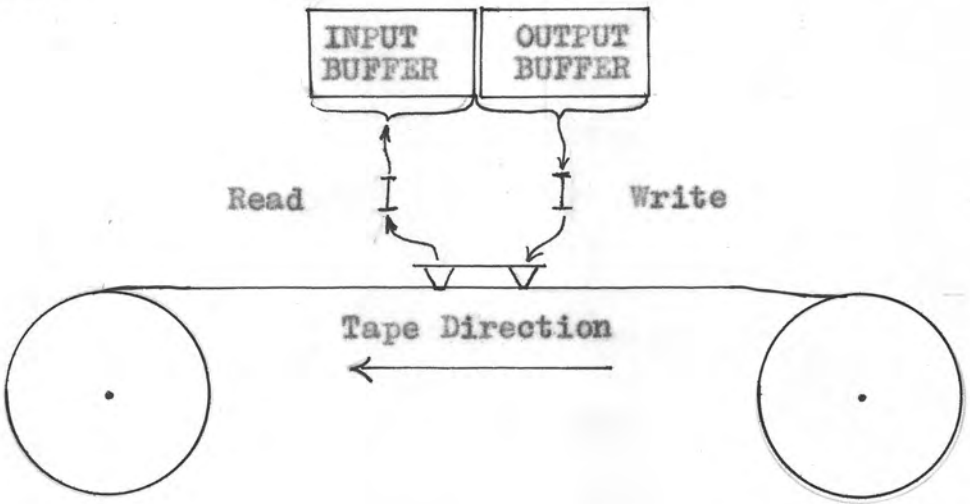


Figure 6.11:- Diagram showing the operation of a magnetic tape unit with a write/read head.

In addition, in the execute phase of a "write" order, the first main store address must be set in both the upper and lower halves of B_{64}.

Obviously, the distances between the read and write heads must be sufficient to allow a minimum delay of 4 words between the two operations. The action taken on finding a discrepancy between what has been read and what should have been written will probably be the setting of another special indicator which the programmer can, if he desires, query with his next order. The sequence of operations following the setting of SI_1 will therefore be as shown in Figure 6.12.

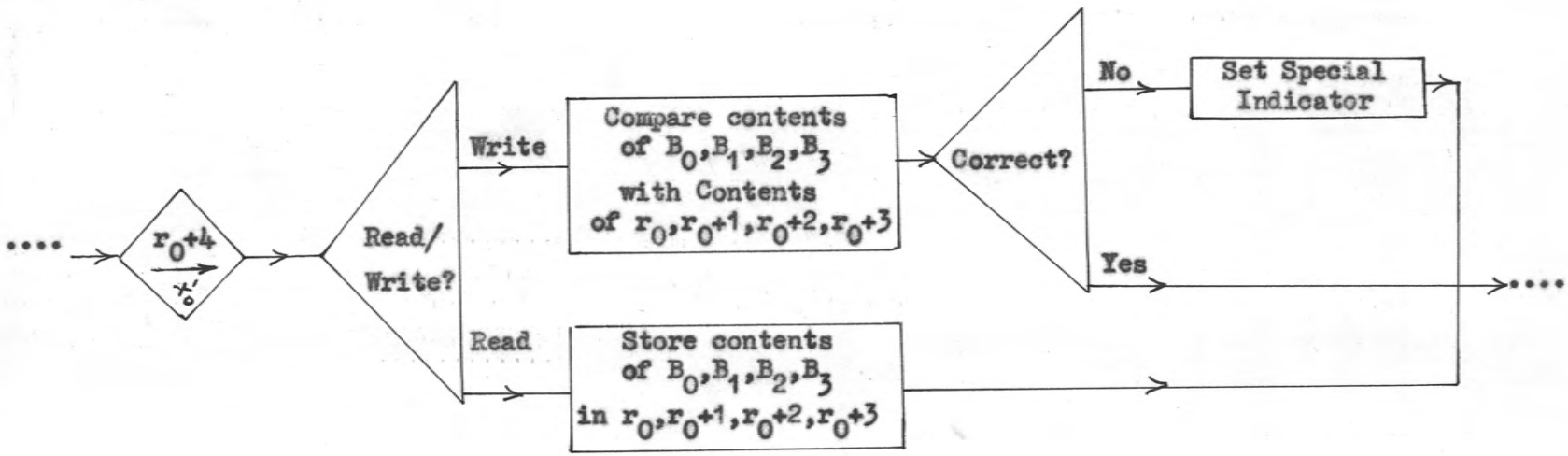


Figure 6.12.:- Diagram of the magnetic tape data handling programme, activated after a change in SI_1. This is one of the interruption handling programmes shown in Figure 6.4.

6.4.1. The Extent to which Simultaneous Operation is Possible.

It is desired that all tape units should be able to operate simultaneously. The most difficult case is when all units are writing, as here there are four writing processes and four processes of reading with checking occurring together. Each transfer between a half-word buffer and the principal buffer will require about 7 μ sec every 198 μ sec., so the operation of eight processes will slow the computer to about 72% of its normal operating speed.

We must allow for the possibility that 8 Secondary Indicators may change at once. The time taken to get into operation a programme corresponding to a Secondary Indicator depends on the position of the Indicator. To minimize this time in the most critical case when all indicators change at once, each of these Secondary Indicator programmes should end by rechecking that $I_1 = 0$. If not, the check of Secondary Indicators may begin with the indicator following the one previously found to be non-zero.

Forgetting for the moment the fact that the computer is on the average for a sequence of operations 28% slower than normal, we can assume that inspecting each secondary indicator will take $2\frac{1}{2}$ μ sec. In this most

critical case, the average number of indicators inspected would be 2; the average delay between finding that I_1 was non-zero and getting the correct Secondary Indicator programme into operation would be about 8 μsec. The Secondary Indicator programmes each require 9 store cycles. Adding the time for checking the read/write indicator, makes the time required with a writing process about 67 μ sec. The reading process would require about 20 μsec. more than this to allow for zero-checks on 8 values.

When four input and four output buffers require attention simultaneously, we must be able to carry out the checks on all the input buffers, refill completely 3 of the 4 output buffers and fill at least the first half-word of the fourth output buffer before the tape unit requires a character from this particular buffer. In computing the time required to pass the critical point we do not consider the time to fill the last buffer completely; we can ignore the last seven store cycle times. The time which elapses before all this is done must always be less than the time spent in writing (4 x 6)+1 = 25 characters. That is to say, 1650 μ sec. Therefore the maximum delay before the computer is ready to begin examining the Secondary Indicators, or, in other words, the maximum time between successive checks of the Primary indicators must satisfy.

$$\tau < t_8 - \left[\sum_{j=1}^{7} T_j' + T_8\right] \qquad \text{.....3.2.}$$

Taking into account the fact that the computer is operating at only 72% of normal speed.

$$\tau < 1650 - \frac{100}{72}\left[4(8 + 87) + 3(8 + 67) + (8 + 18)\right]$$

i.e.

$$\tau < 774\ \mu\text{sec.}$$

This means that the check of Primary Indicators must occur at least once every 774 μ sec. When deciding whether a check of Primary Indicators is necessary other than the one provided in the routine phase for every order, we must consider orders which would, with no magnetic tape units operating, take longer than about .72 x 774 or 547 μ sec.

If these micro-ordered magnetic tape data handling programmes are to include such operations as counting of words in a block, detection of block markers, or searching for particular block numbers, the figures outlined above will be materially affected. In any case, the computed values will require re-working when, in a later stage of the computer's construction, more accurate estimates of its operating speed may be used. However, as we have plenty of latitude in the number of principal

buffer addresses which can be provided, we can make allowances for any reasonable conditions of operation.

7. A CRITICAL ASSESSMENT OF THE CIRRUS MULTIPROGRAMME SYSTEM.

Having put forward a system for achieving multi-programme operation on CIRRUS, it is now necessary to consider to what extent the system satisfies Conclusions 1 - 4 derived earlier, and also how far the system is compatible with the contentions on which those conclusions were based.

Conclusion 1 stated that, if practicable, multi-programme operation was desirable. In preparing the multi-programme system described in this thesis, no insuperable difficulties have appeared, so multi-programme operation is certainly practicable. The assumption of desirability was based on the belief that multi-programme operation would:

(a) Improve the efficiency of operation of input/output equipment.

(b) Obviate operating delays.

As operator delays, under (b) we can also include the inefficient use of the computer in programme testing or maintenance of peripheral units.

Let us attempt to assess how much more efficient, if at all, operation of peripheral equipment might become as a result of using the proposed multi-programme system. Using the estimates of programme switching times made in

Section 6.1.3., let us first estimate how much time during a paper tape reading sequence might be available for use by another programme. Figure 7.1. shows the timing of a fairly typical sequence of operations during reading from paper tape. It has been assumed in this example that the reader operates at 300 characters per second, and that 400 μ sec. is required for decimal to binary conversion of each character.

It will be seen from Figure 7.1., that. instead of having 3,33 m sec between reading one character and reading the next, of which 2.93 m sec are wasted, there will be a minimum time of 3.4 m sec between characters, all but .55 m sec of which is available for use by another programme. The actual period between characters varies, depending on how much time elapses from the moment of the indicator change until the indicators are scanned. From the above figures, it can be seen that there is a net gain in computing time of 2.78 m sec. per character, or about 83% of normal reading time.

The improved efficiency of operation is even more marked during paper tape punching. Assuming a punching rate of 30 characters per second, we have a net gain of 32.78 m sec per character; hence we may use for other programmes about 98% of punching time.

We can estimate the efficiency of magnetic tape

operation from the figures given in Section 6.4. Once every 198 μsec., each tape unit that is reading will require 7 μsec to transfer a half-word to the principal buffer. Each unit that is writing with checking will, over the same period, require 14 μsec. In addition once every 1600 μsec., the unit reading will require 67 μsec for transfers to the main store; and the unit writing will, in the same time, require 67 μsec for transfers from the store and 87 μsec for checking. From this we may say that:

(a) With 4 units writing simultaneously, about 32% of the computer's time will still be available for working.

(b) With 2 units reading and 2 writing simultaneously the proportion of time available for computing improves to about 50%.

(c) With 4 units reading simultaneously, about 69% of the total time will be available for computation.

As a result of the figures quoted above, it may be fairly claimed that multiprogramme operation will considerably improve the efficiency of operation of input/output equipment. It must be remembered also that the degree of efficiency possible will be relatively unaffected by the speed of the peripheral units themselves.

Provided at least one programme in the computer is operable, any delay by an operator will be reduced to

the amount of time involved in two switches of programme. In addition to the appreciable saving in time, this will make operating conditions very much more agreable.

The development of techniques for programme testing on a multiprogramme computer will constitute a complete study in itself. However, it is evident that there is scope for providing CIRRUS with extremely powerful programme checking facilities. Several instruction keys may be used to activate a variety of programmes to aid checking, from complex post-mortem routines to simple programmes such as one which would cause the programme under test to operate until a specified order or address was reached. With the console layout of Figure 6.1., it should be possible for three programmers to test programmes simultaneously, or for one or two programmers to test while working programmes used perhaps 90% of the total computer time.

Conclusion 2 stated that the multiprogramme computer should behave as a set of separate computers. It was felt that this arrangement would be the most effective in minimizing the complications posed by multiprogramme operation. This has to a large extent been achieved as neither the programmer nor the operator need consider any programme other than the one on which they are directly engaged.

READER STATE	TIME SCALE (m.sec.)	PROGRAMME 1	PROGRAMME 2	APPROXIMATE TIME
Available	0	Read character		
Unavailable (Duration: 3.33msec.)		Dec.-bin.conversion		.4msec.
		Programme	switch	.08msec.
	1		General operation	2.85msec. (min.)
	2			
	3			
Available			Programme switch	.07msec.
Unavailable	4	Read character		

Figure 7.1.:- Diagram showing timing of sequence of operations involved in time-sharing during input of characters from paper tape. Net gain per character is 3.33msec. less (.4 + .08 + .07) msec., i.e.2.78msec.

However each must make some small concessions to the fact of multiprogramme operation. In general, the programmer's task will be little different from what it would have been without multiprogramme operation, but he must, for example, observe the restrictions imposed to make store reorganisation possible, and he must go to the trouble of providing the heading for his programme tape. On the credit side for the programmer there is the fact that he need not resort to intricate procedures in an attempt to make use of some of the redundant time during input or output of data.

The operator has to contend with the inconvenience of specifying programme numbers to the Director, and of using those particular peripheral units which the Director allocates. In most other respects, the operator should, in fact, be better off than with the average computer. The necessity for having an operating system more suited to the needs of multiprogramme operation has resulted in the provision of the keyboard and typewriter, which should considerably improve operating conditions. It is likely that programmers will make allowance in their programmes, for directives to be printed on the typewriter at appropriate stages, and as in addition the range of both operating instructions and directives can easily be increased in the light of future experience, CIRRUS should

be an extremely attractive computer to operate.

Conclusion 3 stated that the multiprogramme system should be such that alterations could be made as a result of experience, or to adapt to changed conditions. Virtually any alteration will involve some modification of the Director programme in fixed store. This will not be difficult but it would be unwise, after the computer has been working for some time, to carry out large-scale modifications.

It is unlikely that any circumstances will arise to make it necessary to alter substantially the more fundamental procedures such as the method of using indicators, or of programme switching or the reading of operating instructions. Any modification shown to be desirable through experience is most likely to be an added or altered operating instruction or directive, or an altered layout of peripheral units. Each of these could be easily adopted. Perhaps the most basic amendment which can be foreseen is the possibility that expansion of the system might result in the number of either Primary or Secondary Indicators exceeding 18. If this were to happen, the set of indicators would have to be examined as two separate half-words.

Conclusion 4 stated that provision of a multiprogramme system for CIRRUS should not be allowed to mater-

ially increase its cost. There are certain additional items of hardware which must be directly debited against multiprogramme operation. Neglecting that hardware directly associated with magnetic tape units, we may list these:

(a) Additional peripheral units.

(b) Indicators for peripheral units.

(c) Additional auxiliary registers.

(d) Additional fixed store to allow for the Director programme.

Of these, items (c) and (d) are relatively minor, for they only involve increasing the size of stores which would be provided in any case. The provision of indicators is also not a very costly item. The greatest increase in cost will probably come from the increased number of peripheral units, and this cost will be lessened because slower and cheaper units may be used with almost equal efficiency. Hence, it is evident that the additional cost involved in making multiprogramme operation possible is negligible compared with the benefits which will accrue through using it.

The cost of the hardware associated with the suggested magnetic tape system is of a rather different order. However, it must be remembered that the cost of each tape unit will be about £5,000; adding four of them to CIRRUS

will completely alter our standards of what may be regarded as a reasonable cost. Hence, the cost of this extra hardware is probably moderate when it is considered that, with it, the maximum possible degree of simultaneous operation may be obtained.

In view of all the considerations discussed in this section, the author believes that he is justified in recommending that CIRRUS should incorporate facilities for multiprogramme operation along the lines proposed in this thesis.

BIBLIOGRAPHY.

References quoted in the text:

AEI Electronics Apparatus Division (1960):- "The AEI 1010 Data Processing System - Programming Manual."

Codd E.F., Lowry E.S., McDonough E., Scalzi C.A. (1959): Multiprogramming STRETCH - Feasibility considerations. Comm. Ass. Comp. Mach. Vol. 2., No. 11.

Codd E.F. (1960):- Multiprogramme scheduling. I: Introduction and Theory. Com. Ass. Comp. Mach. Vol. 3 No. 6; II: Scheduling algorithm and external constraints. Comm. Ass. Comp. Mach. Vol 3, No. 7.

Cook R.L. (1960):- Time-sharing on the National-Elliott 802. Comp. J. Vol. 2, No. 4.

Ferranti Ltd. (1960):- "The Ferranti Orion Computer System - Provisional Description."

Gill S. (1958):- Parallel programming. Comp. J. Vol. 1, No. 1.

Grabbe E.M., Ramo S., Wooldridge D.E., (1959):- "Handbook of Automatic Computation and Control." Vol. 2. (John Wiley and Sons Inc.: New York.)

Guidero A.L. (1960):- Computer control of Huntingdon Beach Station. Instrument Soc. Amer. J. (Instrumentation, Systems, Automatic Control) Vol. 7, No. 9.

Harper S.D. (1960):- Automatic parallel processing. Delivered to second conference, Computing and Data Processing Society of Canada, June 6,7, 1960.

Hirst F. (1960):- Melbourne University Computation Laboratory - Report for the period 1-7-59 to 30-6-60.

Strachey C. (1959):- Time-sharing in large, fast computers. Computers and Automation, Aug. 1959.

Wilkes M.V., Willis D.W. (1956):- A magnetic-tape storage system for the EDSAC. Proc. I.E.E. Vol., 103, Part B, p337.

APPENDIX A.

Provisional description of CIRRUS.

STORAGE: (a) Main store. Magnetic core store in blocks of 2,048 words. Initial capacity will probably be 4,096.

(b) Register store. Magnetic core store. Initial capacity will probably be 64 words.

(c) Fixed store. Permanent, wired magnetic core store. Capacity will depend on requirements, but cannot exceed 16,384 words.

WORD LENGTH: 36 bits. Basically, operations will always be performed on 18-bit half-words.

ORDER STRUCTURE: Machine orders will have 36-bits, distributed as shown:

B (6 bits)	Y (6 bits)	F (6 bits)	W (3 bits)	X (15 bits)

F: Function.

W: Word form (18-bit, 36-bit, 72-bit, floating).

B: Register store address (index register)

Y: " " " (accumulator)

X: Main store address $X < 16,384$.

Fixed store address $X \geqslant 16,384$.

SUB-COMPUTER: There will be a simple "sub-computer", comprising the control unit, arithmetic unit and four 18-bit static registers, the M,R,Z,N registers. The sub-computer will be able to perform any of a small set of elementary functions or "micro-orders". The time required to obey each micro-order will vary from 1 μ sec for a control transfer order to about 7 μ sec for an order requiring a complete main store cycle.

The code of machine orders is built up of combinations of the micro-orders, these combinations being permanently wired into the fixed store. Each machine order is obeyed in two parts, the "routine" phase, and then the "execute" phase. The routine phase involves 3 store cycles:

(i) The address of the next order is brought from the main store, increased by one and returned; the expected state of the Primary Indicators is brought from the register store for comparison with the actual state.

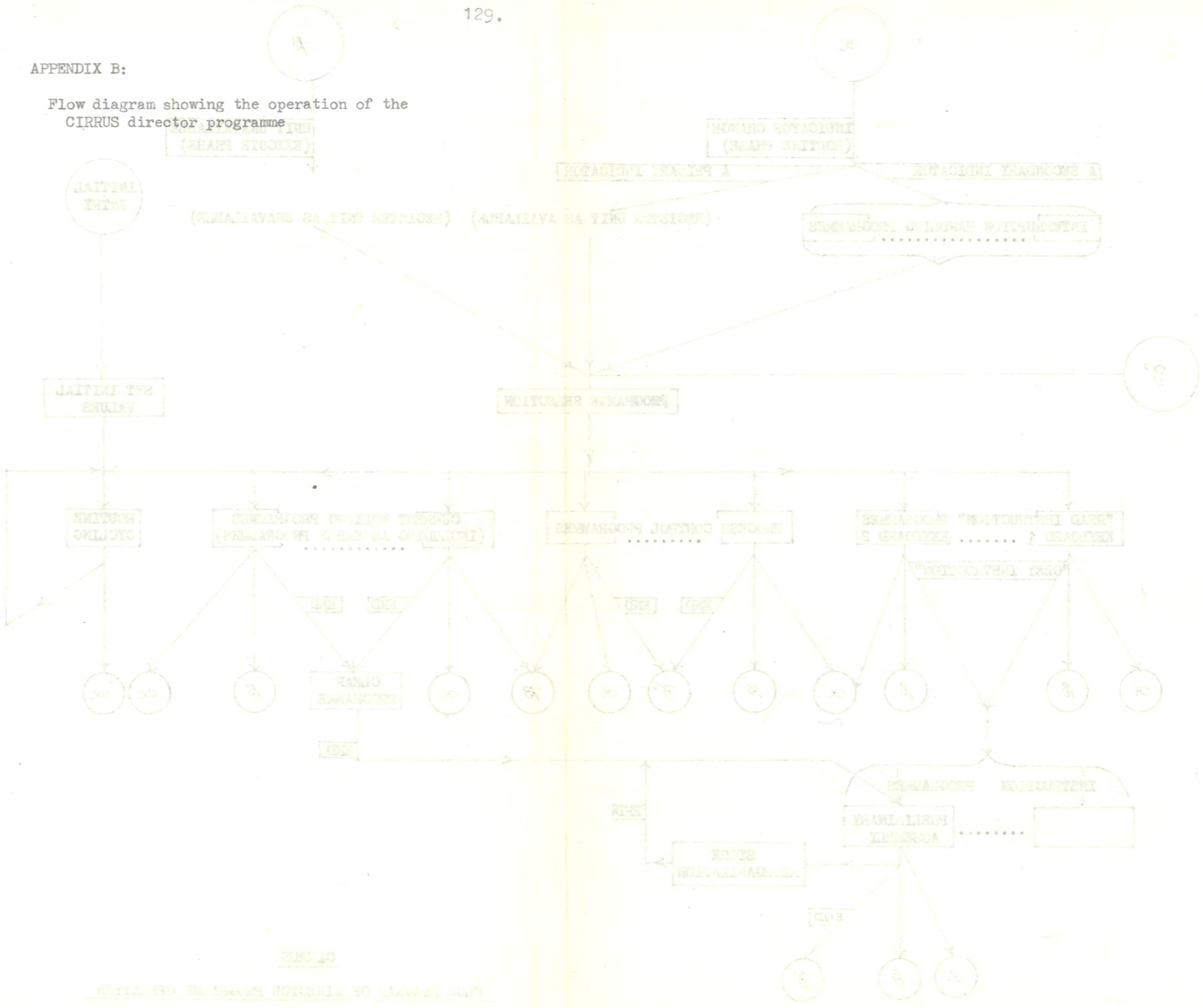
(ii) The first half of the order, containing the function digits and the B and Y addresses is brought from the main store.

(iii) The second half of the order, containing the word form digits and the $\hat{X}$ address is brought from the main store; the contents of the index register are brought from the register store and added to the half-word from the main store.

At the end of the routine phase, control is directed to an execute phase which is selected by reference to the function digits of the order. This execute phase carries out the required operation.

APPENDIX B:

Flow diagram showing the operation of the CIRRUS director programme



α

INDICATOR CHANGE
(ROUTINE PHASE)

A SECONDARY INDICATOR

A PRIMARY INDICATOR

INTERRUPTION HANDLING PROGRAMMES

β

UNIT UNAVAILABLE
(EXECUTE PHASE)

(REGISTER UNIT AS AVAILABLE)

(REGISTER UNIT AS UNAVAILABLE)

INITIAL
ENTRY

γ

PROGRAMME SELECTION

SET INITIAL
VALUES

"READ INSTRUCTION" PROGRAMMES
KEYBOARD 1 ..... KEYBOARD 2

"OBEY INSTRUCTION"

PROCESS CONTROL PROGRAMMES

CURRENT WORKING PROGRAMMES
(INCLUDING ASSEMBLY PROGRAMMES)

ROUTINE
CYCLING

END

CLEAR
PROGRAMME

INSTRUCTION PROGRAMMES

PRELIMINARY
ASSEMBLY

STORE
REORGANIZATION

CIRRUS

FLOW DIAGRAM OF DIRECTOR PROGRAMME OPERATION