This paper outlines a systematic method of designing a data processing tape system utilizing currently available types of equipment.

Primary effort was devoted to obtaining a procedure which would approach an "optimal" system design.

The method presented is an iterative procedure which tends to focus special attention on the critical system functions and the critical relations between functions.

Sequential data processing design

by V. P. Turnburke, Jr.

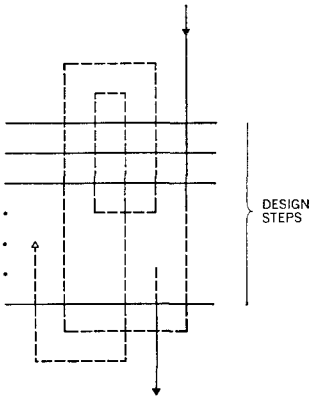
This paper considers the problem of designing a data processing system having a computer equipped with magnetic tape as its principal component. Thus, it is assumed that the system's files are amenable to magnetic tape storage and sequential processing.

In view of the difficulty in formulating a consistent and comprehensive set of design objectives, it is appropriate to emphasize that the initial formulation be subjected to close scrutiny before beginning the actual system design. Even though the objectives may be subject to modification as the design evolves, they should be detailed as completely and accurately as possible at the outset. In addition to functional and economic objectives, those relating to special considerations (e.g., particular requirements such as optical scanning, compatibility with other systems, system backup, cutover schedule, future modifications, etc.) should also be included.

The basic design steps will be outlined in the order in which they are normally undertaken (some overlapping is possible as will be evident). Since each step depends upon earlier steps, at various points it will become apparent that the revision of earlier steps is mandatory or desirable. Thus, the design procedure involves trial and error and has the "iterative" character suggested by Figure 1. Obviously, one cannot guarantee "convergence" to an optimal design. The degree of success will depend largely on the judgment and experience of the systems engineer.

If the system involved random-access memory rather than

Figure 1



method

magnetic tape, essentially the same steps would be applicable, although requiring the evaluation of different parameters.

computer run segmentation

The first step in designing the system is to determine the individual computer runs. Most applications can be divided by function (e.g., input conversion and editing, sorting, master file updating, and output editing and conversion) as a first segmentation.

The division into runs implies a particular configuration which should be consistent with the restrictions imposed by other system needs, such as the minimal requirements for the programming packages to be used, in particular the compiler for the language selected. The selection of the number of components should be economic: for example, one less input-output unit may permit one less control unit.

The segmentation should reflect the requirements for converting source media to magnetic tape and magnetic tape to punched card or printed output. Volumes must be considered in determining whether to convert on-line or by means of a supporting off-line system. This decision may also be affected by special input-output needs (e.g., magnetic ink character recognition of input documents, etc.).

A preliminary run organization chart can now be drawn to show the flow of information through the system and the interaction of runs through common files. This chart will serve in clarifying the scope of the application and the relative magnitudes of the runs.

Segmentation is perhaps the most critical design step since, in addition to its effect on the equipment configuration, the division into computer runs and the consequent interrelation of the runs will influence the total processing time. At several times later in the study, as more information is developed, it may be necessary to review the initial division for possible improvements.

file definition

Each file must be defined in some detail. The source and the destinations on each file need to be established, and any resulting restrictions of file format, such as block size limitations and compatibility requirements have to be identified. Record formats for each file must be defined suitably for both core storage (word size and extra control characters) and magnetic tape (extra control characters and suppression of leading blanks and zeros). The average record length of each file can then be calculated both for tape (in characters and/or digits) and for core storage (in words). Once the record formats are established, the approximate tape passing time for each file can be calculated based on the file volume, the anticipated recording density and a tentative record blocking factor. The approximate number of reels of magnetic tape necessary to hold each file should also be determined at this time.

Each run can be described in terms of input-output requirements. This establishes the overall configuration requirements for the runs as defined by the segmentation. Usually the runs are

defined in descending order of importance, with priority being given to the run which will have the greatest impact upon total running time.

channel assignment

If the computer configuration includes two or more tape channels, the first step for each run is to assign each file used in that run to a channel. The assignments are to be made so that total tape time on the channel with the greatest load is as nearly as possible equal to the sum of the tape times (for all files used in the run) divided by the number of channels.

Usually input files are assigned to one set of channels and output files to another set, but this is by no means necessary. In addition, assignments are usually made with the restriction that each channel will have the same number of tape units, since this is the optimum arrangement for sorting. If this results in a serious conflict, however, it can usually be resolved by adding a tape switching device to the configuration.

As tape files are assigned to channels, special input-output requirements should also be noted. Any file which is contained on more than one reel of magnetic tape should have an alternate tape unit, or else processing must stop while each reel is rewound and changed. The latter choice may be best if the total application requires less than one shift of computer time. If an alternate tape unit is desired, it may be assigned solely to this file or shared with other files. In the case of smaller runs, the decisions made concerning alternate tape units may be influenced by the number of tape units required by other more complex runs.

error and exception routines

Detailed consideration should be given to exception and error routines. Errors detected in the run may have to be collected on a separate tape unit; or they may be combined with other tape output records and isolated in a later run; or punched, printed or typed on-line. The longer runs will require a checkpoint and restart procedure. Checkpoint records must either be assigned to a special tape unit or written on one of the other output files. In the latter case the tape time required by the checkpoint records must be added to the total tape passing time of the file.

adjustment of runs

When the number of tape units required by each run has been established and when tape files have been assigned to channels, a first effort can be made to improve the design. Those runs requiring the most tape units should be examined in an attempt to reduce the number of tape units used. Input files should be traced back to their sources to see whether they can be combined during a previous run into a single file, and output files should be traced to their destinations to see whether they can be combined during the main run and separated in a subsequent run.

Small runs should be examined to determine whether they can be combined to reduce the number of setups necessary. Large runs should be examined to determine whether they can be split into smaller runs, decreasing the size of the configuration required. If any of these steps are taken, the prior steps of cal-

culating file passing time and of assignment of tape files to tape channels must be repeated for the affected files and runs.

detailing the runs

When this initial optimization process has been completed, each run must be laid out in detail. The first step is the calculation of internal processing time and program storage requirements. One element of each which is easily isolated is the Input Output Control System (IOCS) requirement. Both the processing time required and the core storage needed are defined in published manuals as functions of the number of files and the number of tape channels used by the run.

timing estimate

Calculating the balance of the internal processing time which will be required by the program is usually the most difficult problem in designing the system.

Benchmark programming is an accurate method of estimating transaction processing time. It is especially useful if the majority of transactions are of a single type. Note that benchmark programming does not require writing a program which can be keypunched, compiled and demonstrated. There is no need, for example, to consider which conditional branch instruction follows a compare instruction, since they all require the same time and one or the other will work. Exception routines need not be programmed, only the compares to find the active master record, the record moves if any, and the actual master testing and updating routines.

A less accurate approach is to determine the number of macro instructions needed to process a transaction—the moves, compares, adds, tests, etc.—to arrive at an instruction mix for the particular computer. While not as accurate as benchmark programming, this approach at least provides a different mix for each run based on the work which is actually to be done in that run.

Finally, it may be possible to estimate processing time by making a time study of the same run already in operation on the same computer and adjusting the result for differences in record volumes.

Generally, the simple formulas sometimes used for calculating "average" instruction times and "average" numbers of instructions should be avoided because of their inherent inaccuracy.

A special factor, which must be considered as a part of internal processing time, is core-tape interference. This is the processing time used for the transfer of data between core storage and the input-output devices during which the core storage unit is interlocked. It is a function of the total number of words of data read and written during the run and of the computer's design, and is additive to internal processing time.

storage estimate

Program storage requirements include several elements in addition to IOCS requirements which can be defined rather easily. If a supervisory program is to be used, its storage requirements can be allocated based on information from published manuals. An area is usually reserved for the load program which is supplied by the compiler. Storage used by the computer, such as

index words, interrupt words, etc., can be allocated if not already accounted for as part of the load program.

Each file used in the run should be examined to determine whether it should be processed in the input or output area or in a work area. Generally this decision is based on a comparison of the speed of internal data movement to the indexing or address modification time multiplied by the expected number of references to be made to the record. If work areas are to be used, storage should be allocated for them.

If tables of significant size will be needed for reference during the computer run, the format of each table should be established and appropriate storage areas set aside.

The most difficult assessment relates to the storage necessary for program instructions and constants. Again, benchmark programming will provide the best approximation. If a benchmark program has been written to determine internal processing time, it can also be used to estimate program storage by allocating the same amount of storage for each transaction routine required, being sure to adjust the result for error routines and for any transactions more complex than those used in the benchmark program. Alternatively, an existing program written for a similar application might be examined to determine program storage requirements.

Program storage is required for initialization, modifications to the IOCS, and other miscellaneous routines. These routines should be examined to see whether they need be retained in storage at all times, used once and then overlaid by other instructions or data, or maintained on a program tape and read into storage only as required. If necessary, storage should be set aside for these routines.

Additional storage must be allocated to hold program constants, messages, etc. The larger ones, such as printed report headings, can be detailed, and the remainder estimated.

When all of the above storage needs are added together and subtracted from the total, the storage available for input and output areas is known. The number of input-output areas required for each file can be established, based on the system's buffering, the relationships between files (e.g., master file input and output) and the options available in the computer's IOCS.

blocking factors

Blocking factors must now be established for each tape file, based on the available core storage and the number of input-output areas used. Block sizes are optimized when the tape passing time on the channel with the heaviest load is minimized. Since the optimum block size for one run is not necessarily optimum for other runs using the same file, it is best to optimize block sizes in the longest run first and carry these sizes as restrictions for succeeding runs. During the initial iteration of the design process, blocking factors should be optimized without reference to process time. If any run is severely process limited, an attempt should be made to reduce process time.

reviewing the machine configuration

Each of the preceding steps identified certain components so that the configuration is now finalized. This configuration should be reviewed relative to the original objectives. The number and type of magnetic tape units, number of tape channels, size of core storage, and the choice of special features (e.g., process overlap, print storage, high-low-equal compare, etc.) should be examined. Also, the choice between on-line card equipment and an off-line supporting system should be re-examined. If the design objectives have not been attained, the design process to this point must be reiterated until satisfactory results are obtained (or until the problem is found to be without solution).

timing the runs

If the design so far is consistent with objectives, each run must be timed. Utility runs, such as sorts and merges, are timed from published timing formulas. For other runs, tape passing time and internal processing time (including core-tape interference time) must be combined, based on the buffer characteristics of the system. If the system is unbuffered, run time is the sum of tape passing time and internal processing time. For a buffered system, if all internal processing can be overlapped with tape processing, run time is the tape passing time for the channel with the heaviest load. Otherwise, non-overlapped processing time must be added to the latter.

Thus, for buffered systems the non-overlapped internal processing time must be computed. The processing time available per block within tape passing time is in most cases approximately equal to the tape passing time for the channel with the heaviest load divided by the number of master file tape blocks to be processed by the run. For each block, the number of hits (N) that can be processed during tape passing is found by dividing the processing time available per block by the average processing time per transaction (and truncating).

Estimates of non-overlapped processing time obtained by using an average number of hits for each block of the file will usually be highly inaccurate. Computation should be based on the particular distribution of activity.

Taking into account the particular distribution of activity across the master file, for each block the probability of $N + J$ hits within the block, denoted by $P(N + J)$, is computed for $J = 1 \ldots K$, where K is the largest integer giving rise to a significant probability. The non-overlapped processing time for the block, T_b, may be found from

$$T_b = \sum_{J=1}^{K} P(N + J)[T_p(1) + (J - 1)T_p(J)] \tag{1}$$

where

$T_p(1)$ is the non-overlapped processing time generated by the first hit in excess of N, and

$T_p(J)$, for $1 < J \leq K$, is equal to average transaction processing time.

The total non-overlapped processing time can be found by summing the non-overlapped processing time for each block.

Consider the case of an even distribution of activity across the master file in which the probability of a hit against each record may be assumed to be the same. We will further assume, as is typically the case, that the file contains a large number of blocks. (This type of distribution may reasonably represent the activity in certain master files—updating names and addresses, for example.) We may regard each record of a block as involved in an independent binomial experiment with success and failure denoting *a hit* and *no hit,* respectively. Thus, for the entire block, the probability of exactly X hits, if found from the standard formula,[1]

$$P(X) = \frac{G!}{(G - X)!\,X!}(W/V)^X(1 - W/V)^{G-X} \tag{2}$$

where,

G is the number of records in each block,
W is the number of transactions in the transaction file,
V is the number of records in the master file and
W/V is the probability of a hit against an individual record.

Now, values for the $P(N + J)$ can be found from Eq. (2) and inserted in Eq. (1) to permit evaluation of T_b. Since the value of T_b will be the same for each block, to find the total non-overlapped processing time, T, we need only multiply by the number of blocks (V/G) and thus

$$T = (V/G)T_b. \tag{3}$$

Observe that in the above, despite the assumption of an even distribution, it would be undesirable to have based the computation on the average of $(W/V)G$ hits per block. The allowance for individual variation from the average that was actually included in the above computation will give a more accurate result which will usually be significantly different from that obtained by ignoring the variation.

Distributions of the type discussed in the previous paragraph are unlikely in general and a careful analysis of the distribution should be made. Many kinds of master file runs will be found to have a skew distribution. For example, in an inventory master file, 80% of the activity may occur against 20% of the master records, with hundreds of transactions hitting the most active master file record. Accurate timing of such runs requires that the true distribution of activity be determined and considered in the run timing process. The summation of the sums formed by Eq. (2) will still hold. But the computation of $P(X)$ will vary with the block location relative to the distribution of activity. In addition, allowance must be made for multiple hits against a single record. To keep the computation manageable, it will be desirable to divide the entire set of blocks into appropriate sub-

sets and base computation on a "representative" block chosen from each subset. If an excessive amount of non-overlapped process time results, it indicates that a different system design should be sought, one that considers the skew distribution of activity as a parameter of the design.

All run times should next be adjusted for system efficiency. Non-productive time should be added to cover run setup, end of job and any non-overlapped intermediate tape rewinds. The evaluation of setup time should consider whether input files generated by a previous run can remain mounted and the extent to which files required by this run can be mounted on idle alternate tape units during the final minutes of processing of the previous run.

time and cost schedules

When all run time calculations have been completed, the time for each run must be weighted by its frequency to determine monthly run time. The associated costs are calculated and, if consistent with the original objectives, each run should be scheduled by day to assure that no conflict exists between the time an output file is prepared by one run and the time the next run using that file is scheduled to start. The schedule should also consider time elements which are not charged as operating time, but which do affect the total number of hours the system must operate, and therefore affect deadline objectives, personnel costs, etc. Such elements may include setup and idle time. The schedule should verify that peak work loads can actually be accomplished by the system within the required time limits.

The results shown by the cost and operational time schedules should be compared against the original objectives to determine whether the system meets the original design criteria. If one or more of the design objectives are not met, reiteration of certain design steps is clearly necessary. Even if the design criteria are met, it is usually desirable to reiterate in order to find the system which will be most economic for the application—in other words, to optimize the design.

optimizing the design

The first step in optimization is to examine the overall systems design. Assuming that total running time is not so large or so small that a different computer system should be chosen, the most profitable area to examine is that of run segmentation. There are three major approaches to altering the design which offer the possibility of radical changes in cost and running time. The first of these is combining runs. Run combination is effective when the original design results in extra shift computer usage. It may add components to the configuration, depending on the parameters of the runs which are combined. A savings will occur only if the runs being combined use one or more common files, or if a process limited run and a tape limited run can be combined on a buffered system.

The second approach is that of partitioning runs. This is effective when application time is well within prime shift time, and when one or two runs require more components than the others.

Partitioning will add to running time if one or more files must be processed through both runs, unless the run being partitioned is heavily process limited on a buffered system. Reducing the system configuration, however, will usually reduce total cost.

The third approach to optimizing the segmentation of runs is to completely alter the basic system design. The information concerning the scope of the application which was obtained from the initial iteration of the design process allows more objective judgments to be made of each part of the application. Trivial parts of the application should be examined to determine whether they can be done more cheaply manually, in a supporting punched card installation or on a peripheral system. Conversely, it may become apparent that major applications not previously considered can be added with little or no difficulty because the files already described contain the information needed by the new applications. Finally, it may now be obvious that random access provides a better approach than magnetic tape file processing.

individual run design

After the overall system design has been improved, each individual run must be optimized in turn. In general, the longest computer runs should be examined first, since they offer the greatest potential savings. The techniques which can be used to optimize file maintenance runs on buffered systems depend upon whether the run is process limited or tape limited. If the run is process limited, throughput will be increased by reducing process time, if necessary at the expense of tape time. There are three general areas worthy of investigation in optimizing a process limited run. If a previous or a subsequent run is tape limited, it may be possible to move part of the processing to the tape limited run, decreasing the time for the main run without increasing time in the other run. Zero balancing fields in an input transaction and editing fields of an output transaction for printing are obvious examples of process time which can be moved from one run to another without affecting system logic.

It may be also possible to combine all or part of a master file which is processed in a given tape limited run with another master file, reducing or eliminating the given run without proportionately increasing the other. Finally, fields which are required in different formats at different points in the process may be carried in both formats on the master file to avoid the need for editing. This is more useful for information fields such as name, part number, etc., than for numeric fields which are constantly updated.

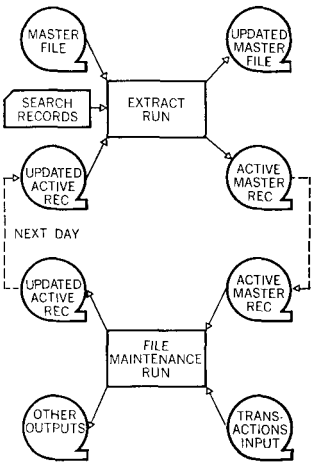
If the run to be optimized is tape limited, essentially opposite approaches can be used to improve it. It may be possible to add units of processing to this run, relieving some other run which is process limited. It may be possible to move master file fields to another, process limited run, perhaps even setting up a second master file for the purpose. Name and address records, for example, might be kept in a special name and address file and extracted on a peripheral system as output documents are being

printed. The master file may be compacted by coding information fields, especially yes-no indicators, in binary. Some master file fields may be completely eliminated, and recreated by programming. If, for example, *Gross Pay — Deductions = Net Pay,* only two of the three fields need appear on the master file.

If master file tape time contributes excessively to the tape limited characteristic of the run, the effect of additional core storage on master file blocking and tape time should be considered. Alternatively, an extraction or a change-tape technique may reduce total job time. An extract run, which may be done either on the main computer or on a peripheral system, searches the master file and extracts the active records for processing, simultaneously filing back the previous day's updated master records (Figure 2).

extraction method

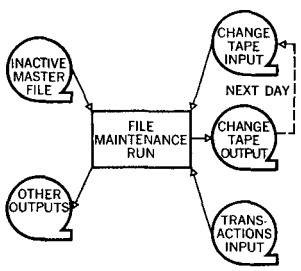
Figure 2



Search records might be either on cards or on tape, and need include only the desired control field. Since the extract program is simple, storage is available to allow a very high blocking of the master file. If the ratio of active master records to total master records is very low, the decrease in master file tape time will more than compensate for the extra reading and writing of active records.

There are many possible variations of the extraction approach, involving schemes which avoid updating the master file daily. One of the most common of these is the change-tape technique (Figure 3).

Figure 3



This arrangement is particularly effective when there is a skew distribution of activity against the master, e.g., when 90% of the transactions hit 10% of the master file items. Essentially, the master file is divided into active and inactive segments, the active segment being called the change-tape. The inactive segment is read only. When a hit occurs on an inactive master file record, it is updated and put in the change-tape. Thus the change-tape content grows, at a rate dependent on the skewness of the activity distribution. As the size of the change-tape approaches one-half the size of the inactive master file, the design begins to lose its effectiveness and a special run is required to move the updated inactive master records from the change-tape back to the inactive master file. This technique is particularly effective on unbuffered or single channel buffered systems.

Since the change-tape approach adds a file to the file maintenance run, it reduces the allowable blocking for each file. If this is critical, the change-tape approach can be used in an extraction run.

split-file approach

If the activity ratio in a master file run is low but each hit requires extensive processing, or if hits tend to occur in bunches that create a process limited situation, run time may be reduced by either a split-file or a queuing approach. In the split-file approach, the master file is divided into upper and lower halves (or thirds, quarters, or even tenths). Transactions must be organized similarly, if necessary on separate tape units. All parts of the file are processed concurrently, so that when activity occurs

on one part, the other parts of the master file can be searched for the next active record while the first hit is being processed. The split-file method is effective only in a buffered system, and increases the number of channel and/or tape units required for the run.

queuing method

Queuing is particularly applicable when the activity is very low but each hit requires extensive processing. Such a run may require nearly as much time on a buffered system as on an unbuffered system. Consider for example, a run designed for a buffered system using two input areas. If a hit requires 200 milliseconds of process time while an input area can be filled by 40 milliseconds of tape time, then each hit will result in 160 milliseconds of non-overlapped process time, and two hits against the same block will cause 360 milliseconds of non-overlapped process time. This non-overlapped process time occurs because there is only one empty input area at the time of the hit. If the number of input areas is increased to allow a queue of unprocessed data to form when a hit occurs, the non-overlapped process time is decreased. In the above example, if six input areas were used, five of them could be filled with a queue when a hit occurred and non-overlapped process time would drop to zero on one hit, and to 200 milliseconds on two hits. Depending upon the parameters of the job, it may be possible to achieve a similar result without using additional core storage by decreasing tape blocking. In the same example, dividing the original core storage into six input areas instead of two would decrease blocking by a factor of three and increase tape time by about 20%. If non-overlapped process time were 30% or more of total run time, however, run time would decrease. Note too, that the lower blocking factor will reduce the probability of more than one hit per block.

scheduling and system size

If the last iteration of the design resulted in extra shift usage, then additional core storage, tape channels and tape units should be considered. The addition of components will generally have its greatest impact on sorting times, so the sorts should be reevaluated first. On the other hand, if the previous design was well within one shift it may be possible to remove components from the system, increasing running time but reducing total cost. Here too the greatest impact will be on sorting times, though other runs probably will have to be redesigned for the smaller configuration.

Careful consideration must be given to the frequency with which runs are performed. If the application includes weekly runs, it is usually more efficient to perform a single preliminary sort or merge. If, however, the weekly runs cause an excessive end-of-week load, the scheduling problem may be alleviated by sorting daily and providing for five inputs to the following run, merging internally rather than within a special run. The possibility of dividing a weekly file maintenance run into five daily runs, each processing one-fifth of the file, should be evaluated as an alternate approach.

sorting

There are several ways to optimize sorting times and costs. Each sort should be timed for the next lower order of merge, for

a higher order of merge will not improve running time unless it decreases the number of merging passes required. If Phase I of a sort is process limited, it is possible to reduce the length of the strings produced by Phase I, reducing processing time. This will increase the number of strings generated by Phase I but will not appreciably affect Phase II time until the number of strings becomes large enough to force an additional Phase II pass. In most utility sort programs the control card can be used to reduce the storage available for sort areas in Phase I, thereby reducing the string length.

Large volume sorts always suggest the possibility of writing a special sort program rather than relying on utility sorts. Programming of specialized sorts has been greatly simplified by the availability of IOCS subroutines, for most sorting techniques are essentially input-output problems. Digital sorting is an obvious approach for a specialized sort if the records have short numeric control fields. It may be similarly effective if there is an inherent bias to the control fields which allows some form of block sorting which takes advantage of the skewed distribution of digits in each control field position or groups of positions.

Sort time may also be decreased by ordering each group on the sort input tape in the previous run and bypassing Phase I of the sort run altogether. This approach may reduce sort running time even if an additional Phase II pass is required, because a Phase II pass is usually faster than a Phase I pass. It is especially effective if a high degree of order already exists in the file to be sorted.

concluding remarks

Of course, there are a myriad of additional approaches in addition to those itemized above. When all promising alternatives have been evaluated, it is useful to look at the overall system design and ask the following questions:

1. Have all objectives been attained?
2. What are its strong points? weak points?
3. Where is the time and cost concentrated?
4. Is there a radically different approach that should be examined?

If the answers to all these questions are satisfactory, the system may still not be the very best possible, but it should be a good solution to the design problem.

ACKNOWLEDGMENT

The author wishes to express appreciation to his colleagues, F. S. Beckman, J. Svigals, and P. T. Woitach for their generous assistance.

FOOTNOTE

1. This formula is derived and tabulated in the standard statistical texts. See for example, Mosteller, Rourke and Thomas, *Probability with Statistical Applications,* Addison-Wesley Publishing Co., 1961.