Technical Report TR-147
NGR21-002-197

January 1971

Evaluation of an Adaptive Approach to Buffer Storage Allocation

by
Sarah Crooke
Jack Minker

UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER
COLLEGE PARK, MARYLAND

Technical Report TR-147 January 1971
NGR21-002-197

Evaluation of an Adaptive Approach to Buffer Storage Allocation

by
Sarah Crooke
Jack Minker

This research was supported in part by Grant NGR21-002-197 from the National Aeronautics and Space Administration to the Computer Science Center of the University of Maryland.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

1. Introduction

This report investigates an adaptive approach to the allocation and release of buffer storage within a system. Specifically, three techniques are evaluated. These are the buddy method, the first-fit method, and an adaptive method which uses the buddy system and the first-fit system depending on the user input statistics. The adaptive approach uses a predictor which determines when statistics change indicating when one should move from the buddy to the first-fit system and vice-versa. The approach to using the adaptive method was motivated by the fact that the buddy system was effective with respect to serving small buffer request sizes while the first-fit method was most effective for the allocation of large buffer storage. Investigation, therefore, considers the case when one is dealing with a system where the statistical characteristic of the buffer request sizes interchanges between small and large requests.

1.1 Scope of Study Reported in This Paper

This report is a study of an adaptive approach to the dynamic allocation of buffer storage. Three major facets to the effort are discussed: a) implementing an adaptive approach which is controlled automatically as a function of the user statistical environment, b) reducing inefficiencies caused by the interaction of the individual strategies used in the adaptive mode, and c) arriving at a measure suitable for evaluating the performance of the individual strategies as well as the adaptive method.

1.2 Adaptive Method Control

The actual use of an adaptive approach is dependent on the user statistical inputs. Depending upon the statistics, one method may be

preferable to another, and we may desire to change from using one algorithm to using a different and perhaps more effective algorithm. Therefore, provision for internal monitoring of the statistical environment is required to determine when the alternate strategies in the adaptive method should be interchanged. In the dynamic allocation of buffer storage, the average buffer request size and the existence of queued requests are the parameters of interest. In the simulation models, these parameters are determined at the end of each processing interval. The criteria for alternating strategies in the adaptive mode are based on the current average buffer request, the change in the average buffer request over the last interval, and the existence of queued requests.

For comparison purposes, two simulation models were developed. The first simulation runs were made with perfect prediction. In this model, use of the alternate algorithms in the adaptive method was controlled so that if the request distributions used to generate the buffer requests in the simulation were interchanged, the allocation algorithms were interchanged at the same time. Simulation runs were then made using the second model where the use of the allocation algorithm was controlled based on the monitored average buffer request and the existence of queued requests. Here, the decision to interchange algorithms is based on a threshold value and data obtained from internal monitoring of the allocation process.

This particular aspect of the study is essential to determine the practicality and feasibility of employing an adaptive strategy. Specifically, one must be assured that effective control of the allocation process can be obtained through the selection and the implementation of function monitors, and that reliable control of the allocation process

can be maintained through the use of proper control criteria.

1.3 Side Effects of Adaptive Strategy

In considering an adaptive strategy, one of the basic problems is that of making two independent algorithms compatible. This involves devising a mechanism which will permit transition from either algorithm to the other as required. If it is found that the individual strategies, their data structures, and their inherent properties permit interchanging them, the effect of alternating the strategies must be investigated.

In the simulation of the adaptive strategy for the dynamic allocation of buffer storage which is being investigated here, it was noted that whenever a transition was made from the first-fit method to the buddy method, there was a significant increase in the number of small buffers placed on the available buffer lists. Some analysis was undertaken to determine the factors giving rise to this situation. An attempt was then made to modify the transition mechanism in an effort to either reduce or eliminate this phenomenon. The simulation model was then modified and the results were compared.

It was clear from the simulation results that the problem had not been eliminated. Further it could not be eliminated, only reduced. The problem lay not with the transition mechanism itself, but rather in the basic incompatibility of the two allocation methods. The buddy method requires that every buffer size be a power of two and have a proper starting location. The first-fit method insures only that any buffer allocated is a multiple of four. It is clear then that the greatest number of buffers allocated by the first-fit method will not be a power of two. In the release process, prior to use by the buddy method,

buffers if allocated by the first-fit method, must be checked and split as required to insure a valid size and start address. As a result, an increase of small available buffers was introduced.

1.4 Evaluation of Allocation Performance

Obtaining a valid measure of performance is essential. One would like to compare the performance of the individual allocation strategies, the adaptive strategy, and any modifications to these strategies. Further, in order to attain one of the objectives of this research, i.e., to determine the effectiveness of the internal monitoring and the criteria used to control the use of the individual algorithms in the adaptive strategy, a measure of performance is essential.

Two factors, time and storage utilization, were considered initially. Since in the simulation process algorithm execution times were not available, a search was made for a measure of storage utilization. Two types of memory loss are present - internal and external memory loss. In either type, the memory loss is that part of memory which is available but unusable to satisfy a buffer request.

Denning [5] defines these two types of fragmentation in terms of the conditions under which they occur as follows. External fragmentation occurs when it is found that for a given buffer request, every available buffer is too small to be used; or, external fragmentation occurs for requests of size s with some probability E(s), where E(s) is the probability that $s > \max\{x_i\}$, where x_i are the available buffer sizes. Internal fragmentation occurs in cases where the buffer sizes which are acceptable are restricted because storage requests must be rounded up to the next acceptable buffer size and the difference between the buffer size allocated and the buffer size requested is lost. More precisely,

if $\{z_i\}$ is the set of acceptable buffer sizes with z_i arranged in ascending order and s is the size of the request, if $z_{i-1} \leq s < z_i$, then $z_i - s$ words are wasted inside the buffer allocated.

It was found that internal memory fragmentation can be monitored and measured quantitatively. Attempts to measure external fragmentation based on the number of buffers on the available storage list or words represented by those buffers were unsuccessful. A large number of small buffers on the available lists is unimportant if it is found that queues of unhonored requests are never present. The length and duration of queued requests as a basis for measuring external fragmentation alone was unsatisfactory since the formation of queued requests may result from either internal or external fragmentation or more often a combination of the two in the adaptive method.

The probability that a request for a buffer of size x will be queued was the measure finally adopted because of difficulties determining the relative importance of internal and external fragmentation. In this measure, two factors were used: the probability that there would be a request for a buffer of size x and the probability that the request could not be satisfied. Since the measure reflects the effect of total fragmentation it is useful in comparing the performance of the allocation strategies regardless of the type of fragmentation, internal or external, which is present in the allocation process.

2. Review of Previous Efforts

The concept of an adaptive strategy for dynamic buffer storage allocation as described in this paper was first introduced in [1], and explored more extensively in [2]. These two studies were directed toward analyzing strategies for the dynamic allocation of buffer storage for temporary, unpredictable, and small buffer requests. The analysis made use of results obtained from simulation models and used some data which were obtained from internal monitoring of the EXEC 8 operating system for the Univac 1108 at the University of Maryland.

Briefly, two basic strategies for handling the dynamic allocation of buffer storage were modeled. The first strategy modeled was the buddy method. This method was first used by H. Markowitz in connection with the SIMSCRIPT programming system [3] in 1963. The second method of allocation modeled was the first-fit method. Algorithms for these two basic strategies along with limited results from simulation models are given in Knuth [4].

2.1 Buddy Method

The characteristic feature of the buddy allocation method is that regardless of the exact buffer size requested, a buffer of size 2^k is allocated, where k is the least power of 2 which is greater than or equal to the buffer size requested plus one word of overhead. Although a request may be made for any size buffer within a specified range, the size of the allocated buffer is always a power of two, representing essentially a restricted number of distinct buffer request sizes. As a consequence, the lists of available storage buffers are maintained by size.

Circular lists, singly linked are used for storing available blocks of storage, with one word of overhead in each allocated block used for

allocation control. If 2^m is the largest buffer size permitted, then (m-1) locations are used to serve respectively as heads of the lists of available buffer lists of sizes $2^2, 2^3, \cdots 2^m$.

2.2 First-Fit Method

The basic characteristic of this method of allocation is that allocation is made from the first available buffer found which is greater than or equal to the size requested. The algorithm modeled for this study maintains the same number and structure of lists as found in the buddy method. In the allocation process, if the difference between the buffer size requested and the available buffer from which the allocation was made is less than four words, the whole block is allocated. This avoids the possibility of returning a block to the available list which is so small that it is virtually useless in satisfying future requests. Further, all buffers allocated and all buffers maintained on the available lists are a multiple of four, and if a buffer of size n is available it is placed on the $(i+1)^{th}$ list where $2^i < n \leq 2^{i+1}$.

2.3 Results from Buddy and First-Fit Models

In the work reported on in [1,2], the performance of the two allocation methods was measured in terms of execution time and memory utilization. The decision to implement a particular strategy is one which the system designer, or analyst, must make, based on the premium placed on time or space.

In order to estimate relative execution times, data were obtained on the time consuming operations within the allocation processes. These included the number of searches of the available buffer lists needed to honor a request and the number of possible memory collapses upon return

of a buffer. As noted by Knuth [4] and as further shown in [2], time-wise, the buddy method was found to be faster than the first-fit method.

Efficient memory utilization involves minimizing two types of memory waste: internal and external fragmentation. Internal fragmentation may be introduced whenever the size of the buffers allocated is either fixed or restricted to a limited number of specified sizes. This type of memory fragmentation is almost unavoidable if the buddy allocation scheme is used since the size of any buffer allocated must be a power of two regardless of the actual buffer size needed. The first-fit method as modeled for this study also introduces some internal fragmentation since buffers allocated are always a multiple of four. In the buddy method the memory loss may be as large as one half 2^m, where 2^m is the largest buffer permitted, while the first-fit method insures that the internal memory loss per allocation will always be less than four. Based on the simulation models and internal fragmentation only, it was found that whenever the average request size is greater than four times the average overhead of the first-fit method, more memory is required by the buddy allocation scheme than by the first-fit method [2].

As noted by Randell [6], external fragmentation is introduced when the acceptable buffer requests are unrestricted as to size. If external fragmentation is present, it is highly improbable that all available space will be used before apparent overflow occurs in the allocation process. Apparent overflow results when a request is made for a buffer of size n and this request cannot be honored; however, the difference between the total memory reserved and the total memory allocated is greater than n. In such cases, it is clear that if the allocated buffers were placed contiguously in the memory pool, n consecutive locations

would be available to satisfy the buffer request.

External fragmentation was not measured quantitatively. From simulation results and memory maps constructed from EXPOOL in the EXEC 8, it appears that this type of fragmentation is minimal when using the buddy method. Knuth [4] notes that in his simulation of the buddy method, 'in cases where apparent memory overflow occurred, memory was 95% packed and this reflects a surprisingly good allocation balance'. The external fragmentation becomes a much more serious problem when using the first-fit method since the block sizes requested are unrestricted.

The conclusions drawn in [2] based on a study of the above two methods of allocation may be summarized as follows:

a) If allocation time is the only criterion for selecting an algorithm, the buddy method is superior to the first-fit method since the operations involved in the allocation and release processes are most efficient.

b) Internal fragmentation introduced by the buddy method may be unacceptable when the average buffer size requested is large. External fragmentation introduced by the buddy method is minimal.

c) Internal fragmentation introduced through the use of the first-fit method may be eliminated or as in this study, have an upper limit imposed. External fragmentation caused by the first-fit method may be a serious problem because of the number of request sizes permitted and the long term memory checkerboarding effect produced.

2.4 Adaptive Method

The results presented above indicate that the type and severity

of memory fragmentation in each allocation method differ and are a function of the user input statistics, in particular, the user request and release distribution for buffer storage. If the average request size is large, internal fragmentation introduced by the buddy method may be unacceptable; however, the buddy method is most efficient if small buffers are predominant in the request distribution. In view of the possibility of change in the request distribution over time or with different modes of operation, an adaptive approach was investigated.

The implementation of an adaptive scheme in a real operating system depends on the solution of two problems. The first involves providing a mechanism for automatically replacing one algorithm by the other without interruption to the allocation process. The second involves selecting criteria which accurately reflect change or rate of change of conditions in the user input environment and providing a monitoring device which detects and signals the occurrence and direction of any significant change.

Using the buddy method and the first-fit method of buffer allocation, the means of providing for an automatic transition from one algorithm to the other was found. The adaptive strategy was modeled and simulation results were obtained. (See Chapter IV [2]). Change from one algorithm to the other was controlled in the work reported in [2]. Two request distributions were used and requests were generated using these distributions. When one distribution used to generated buffer requests was replaced by the other, the allocation algorithms were interchanged. The change from one algorithm to the other was in effect based on perfect knowledge of when the user distributions were interchanged.

In a dynamic situation, such knowledge is unrealistic. It is

expected that the change from one distribution to another is gradual or even if abrupt, in general no foreknowledge is available as to the time of occurrence. For these reasons some monitoring of the request distribution is needed which can form the basis for deciding which algorithm should be used during each time interval.

3. Analysis of Adaptive Strategy Implementation and Performance Evaluation

In this section a detailed description is given of the analysis performed in implementing and evaluating an adaptive strategy for the dynamic allocation of buffer storage. The total analysis can best be described in three parts. The first is concerned with selecting parameters which could be used to provide reliable control of the use of the individual allocation algorithms in the adaptive mode. In the second, the interaction of the individual allocation algorithms when used in the adaptive strategy is investigated in an effort to improve allocation performance. The third part of the analysis is concerned with establishing a measure which permits the allocation methods to be compared.

3.1 Parameter Monitoring and Control of the Adaptive Strategy

The objective of this aspect of the analysis was to select parameters which could be monitored easily during the allocation process and which could be used to provide a basis for control of the alternative allocation algorithms in the adaptive mode. In [2] the efficient storage utilization of the first-fit and buddy methods was found to be a function of the average buffer request size. Therefore, the monitored value of the average buffer request size and the change in the average request size were selected as the basic elements in the formula used to predict when allocation strategies should be employed.

The general form of the prediction is as follows:

(3-1) Predicted Value of $\bar{x} = (x_p) = \bar{x} + \rho . \dot{\bar{x}}$

where $\bar{x}$ is the average buffer request size, $\dot{\bar{x}}$ is the buffer request change over the processing interval, and ρ is an adjustable parameter used to predict change in the succeeding interval based on the change in

the current interval.

Two other factors, the existence of queued requests and a threshold value, are used in conjunction with the predicted value. The existence of unhonored requests was found to be important in the results reported in [2] and a threshold value used is that average request size at which one allocation method becomes less efficient while the other becomes more efficient depending upon the direction of change of the request size. Therefore, whenever the predicted value exceeds or falls below the selected threshold value for two successive processing intervals, the allocation algorithms best suited to allocate those average buffer request sizes is introduced. Two allocation intervals are chosen so that spurious changes do not require changing strategies.

Specifically, if allocation is being made using the first-fit method and the predicted value is less than the threshold value for two successive processing intervals and there are no queued requests, the buddy allocation method is introduced. If queues exist, no change is made regardless of the predicted average request size. If the buddy method of allocation is in use, either of two conditions may cause the first-fit method to be introduced. Either a queue of unhonored requests is formed, in which case a change is made to the first-fit method of allocation at the end of that processing interval; or no queue exists. In the latter case if the predicted value is greater than the threshold value for two successive intervals, the first-fit method is again employed.

3.1.1 Control of Adaptive Method

As indicated above, the two parameters considered most critical

in controlling the use of the adaptive strategy were the average request size and the formation of queues of unhonored requests. In an actual operating system, more memory would be allocated to the buffer pool so that queues would not be present, or, as in the EXEC 8 system, a sensitive mode of operation could be entered where only priority requests are honored until sufficient buffers have been released to resume normal operation. In the simulation process, enlarging the memory pool was not possible, so unhonored requests were queued and the existence of queued requests is included as a factor in the control of the adaptive strategy.

In [2] it was found that the first-fit method of allocation reduces queued requests more quickly than the buddy method. This is particularly true if queues are formed and a change is made from the first-fit to the buddy method. In this case, the buffers being released were allocated by the first-fit method and, in general, are not the correct size for the buddy method. Therefore, in the process of returning them to the available list, many large buffers are split by the buddy method. This in effect reduces the number of large buffers available to honor large requests which were queued.

In view of the above results, if a queue exists, it was decided that no change from the first-fit to the buddy method should be made even if the request size were favorable for using the buddy method. When queues no longer exist, the buddy method may be introduced as a function of the request size. On the other hand, if queues form while allocating using the buddy method, a change is made to the first-fit method since the first-fit method introduces less internal fragmentation and is more economical of the limited space available.

3.1.2 Simulation Performed

Although the adaptive strategy allocation and release execution times can not be obtained from the simulation performed, execution time was considered in deciding on the internal parameter monitoring to be used with the adaptive method. First, the interval over which requests are averaged is determined by the number of requests made and a power of two was selected as the number of requests which should define an interval. This permits obtaining the average request size for an interval with one computer operation, a binary shift, as opposed to first computing a time interval, and then dividing the request sum by this number. The latter not only increases the computation time, but, if average request size is required, additional computation must be performed. Second an attempt is made to select an interval long enough so that the frequency of computing the average request size is low and so that false predictions are not made that cause rapid oscillations between the alternative strategies. Such transitions result in increased running time and inefficient allocation.

If the interval is too long, significant change in the request size may be averaged out so that the change is undetected. Even if the change is detected, introduction of the alternative strategy may be delayed longer than desirable. Using the rate of change of the average buffer request and the current average value of the request size, the predicted average for the next interval is calculated as a linear combination of the two estimates. At the end of the following interval, the calculation is again performed.

In the simulation, the interval is determined by 64 buffer requests. First, the average request size for an interval, $\bar{x}_i$, and the

change in request size, $\dot{\bar{x}}_i$, are computed where:

$$(3\text{-}2) \qquad \bar{x}_i = \sum_{j=1}^{64} x_j/64$$

and,

$$(3\text{-}3) \qquad \dot{\bar{x}} = (\bar{x}_i - \bar{x}_{i-1}).$$

The prediction for the following interval is then based on the current request size, $\bar{x}_i$, and the change from the previous interval, $\dot{\bar{x}}_i$, as follows:

$$(3\text{-}4) \qquad \text{predicted } \bar{x}_{i+1} = \bar{x}_i + \rho(\bar{x}_i - \bar{x}_{i-1})$$

where ρ is a parameter which may be adjusted to provide reliable prediction. Here again, it is recommended that this factor be a power of two to take advantage of the binary shift operation. In this study ρ was set at 2^{-1}. If the requirement that the prediction should be greater than the threshold for two consecutive intervals were not imposed, this value for ρ would lead to spurious signals to change strategies. See Table III-1. However, with the two interval requirement and this setting of ρ, acceptable control is maintained.

The simulation was performed using two buffer request distributions, d_1 and d_2, with an average buffer request size of $\bar{d}_1 = 15.2$ and $\bar{d}_2 = 20.3$ words respectively. See Figure III-1. The selection of these request distributions was based on two factors. First, the shape of the distributions is realistic in view of the request distribution found in the Univac 1108 EXEC 8 system. Second, the average buffer request sizes represented by these distributions are distinct and satisfy necessary conditions for testing the predictor simulated in the adaptive strategy.

The distributions were used alternately for significant periods

of time to generate the buffer requests. In Figure III-2, the average request size per interval and the distribution in use are shown. Two series of simulation runs were performed to determine the effectiveness of the prediction and control method described above. In the first, 'perfect prediction' of the buffer request size was used to control the use of the alternative allocation strategies. In this mode, when the request distribution was changed, the allocation method was changed. In the second, the predictor mechanism was used at the end of each interval to determine whether the allocation method should be changed.

3.1.3 Simulation Results

The results show that the first-fit method was in use more in the predicted control than when control was based only on the request distribution in use. This is explained by observing that control in the predictor method is based on the existence of queued requests as well as the average buffer request size whereas the perfect prediction was based only on the average request size, i.e., the distribution used in generating the requests. As will be seen later when the allocation performance of the various methods are compared, the effect of using the buddy method while queued requests exist is significant.

The results of the simulation indicate that the predictor method of controlling the adaptive method describe above is feasible. Based on results it is concluded that if the adaptive strategy were implemented in an actual operating system with similar user input statistics, adequate control could be maintained using this method. Further, the parameter monitoring is minimal and the prediction calculation is uncomplicated since it is simply a linear combination of the control

Interval No.		$\bar{x}$	$\Delta\bar{x}$	$\rho.\Delta\bar{x}$	Pred. $\bar{x}$
39	1	19.77			**
	2	15.06	-4.71	-2.36	12.70
	3	19.34	+4.28	+2.14	21.20
	4	15.52	-3.82	-1.91	13.61
	5	15.14	- .38	- .19	14.95*
	6	21.48	+6.34	+3.17	25.65
	7	15.72	-5.76	-2.88	12.84
	8	14.69	-1.03	- .52	14.17
	9	16.55	+1.86	+ .93	17.48
	10	16.00	- .55	- .28	15.72
	11	20.88	+4.88	+2.44	23.32
	12	16.20	-4.68	-2.34	13.86
	13	17.33	+1.13	+ .57	17.90
	14	16.04	-1.29	- .65	15.39
	15	20.19	+4.15	+2.08	22.27
	16	12.42	-7.77	-3.89	8.53
	17	14.20	+1.82	+ .91	15.11**
	18	17.89	+3.69	+1.85	19.74
	19	28.06	+10.17	+5.08	33.14*
	20	18.41	-9.65	-4.88	13.58
	21	25.80	+3.39	+1.70	27.50
	22	20.97	-4.83	-2.42	18.55
	23	16.83	-4.14	-2.07	14.76
	24	21.58	+4.75	+2.38	23.96
	25	15.48	-6.10	-3.05	12.43
	26	18.05	+2.57	+1.29	19.34
	27	18.92	+ .87	+ .44	19.36
	28	19.42	+ .50	+ .25	19.67
	29	20.42	+1.00	+ .50	20.92
	30	20.38	- .04	- .02	20.36
	31	22.72	+1.34	+ .67	23.39
	32	16.76	-5.96	-2.98	13.78**
	33	13.36*	-3.40	-1.70	11.66*
	34	15.42	+2.06	+1.03	16.45
	35	11.56	-3.86	-1.93	9.63
	36	13.50	+1.94	+ .97	14.47
	37	17.30	+3.80	+1.90	19.20
	38	13.50	-3.80	-1.90	11.60
	39	17.30	+3.80	+1.90	19.20
	40	12.94	-4.36	-2.18	10.76
	41	19.06	+6.12	+3.06	22.12
	42	13.81	-5.25	-2.63	11.18
	43	17.05	+3.24	+1.62	18.67
	44	17.52	+ .50	+ .25	17.77
	45	13.66	-3.86	-1.93	11.73
	46	12.88	- .78	- .39	12.49
	47	20.40	+7.52	+3.76	24.16
	48	12.13	-8.27	-4.14	7.99
	49	20.33	+8.20	+4.10	24.43
	50	14.63	-5.70	-2.85	11.78

TABLE III-1. COMPARISON OF PREDICTOR CONTROL AND PERFECT PREDICTION BASED ON REQUEST DISTRIBUTION CHANGE.

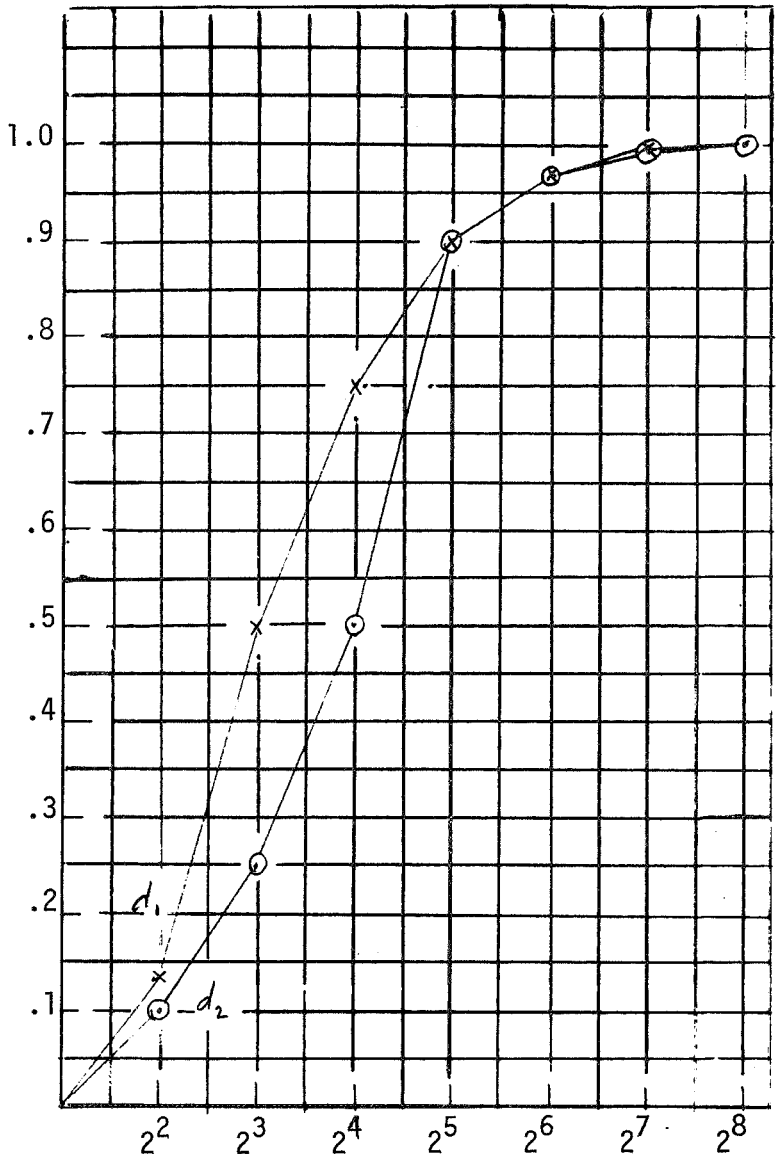
TABLE III-1. (Continued)

Pred. $\bar{x}_{i+1} = \bar{x}_i + \rho\Delta\bar{x}_i$

*denotes when change should occur based on predictor

**denotes when distribution d_1 and d_2 were interchanged when $\bar{x}_{d_1} = 15.18$ and $\bar{x}_{d_2} = 20.25$

TABLE III-1. COMPARISON OF PREDICTOR CONTROL AND PERFECT PREDICTION BASED ON REQUEST DISTRIBUTION CHANGE.



AVG. REQ.

$\overline{d_1}$ = 15.18

$\overline{d_2}$ = 20.25

AVG. ALLOC (BUDDY)

$\overline{d_1}$ = 21.24

$\overline{d_2}$ = 27.00

PROBABILITY OF REQUEST SIZE

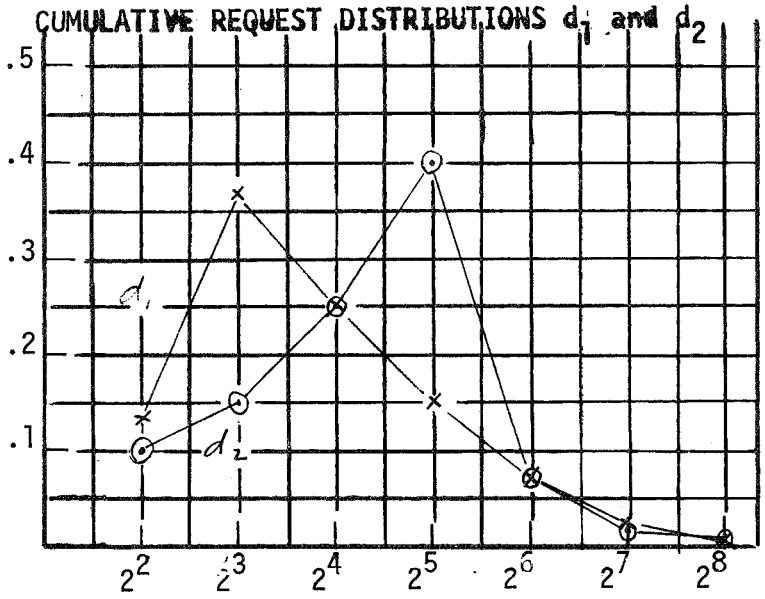


FIGURE III-1. BUFFER REQUEST DISTRIBUTIONS d_1 and d_2

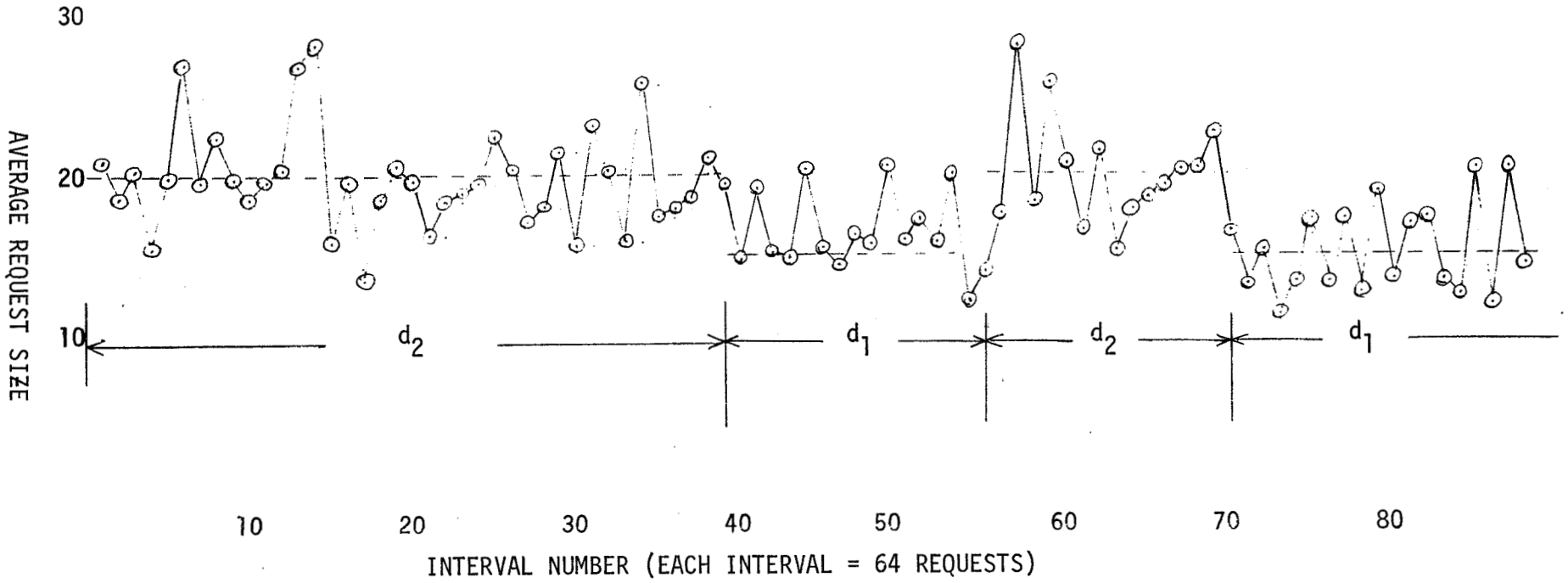


FIGURE III-2. AVERAGE REQUEST SIZE PER INTERVAL AND DISTRIBUTION d_1 or d_2 (WHERE $\bar{x}_{d_1}$ = 15.18 AND $\bar{x}_{d_2}$ = 20.25)

parameters. It should be noted that with other user statistical environments, the predictor could become more complex through the introduction of higher order approximations. However, it appears that the same parameters, the average buffer request size, the change in request size, and queue formation, would be involved in controlling the adaptive strategy.

3.2 Observed Side-Effects in Adaptive Strategy

Before arriving at the method for comparing the allocation strategies described in Section 3.3 of this report, several other methods were considered. Among these was one which attempted to estimate external fragmentation, where external fragmentation was based on the number of small buffers on the available buffer lists. It was noticed here that when a transition was made from the first-fit method of allocation to the buddy method that there was a surprising increase in the number of buffers on the buddy available lists. This was surprising since in general when the buddy method is used throughout there are fewer available buffers than when the first-fit is used throughout. See Figure III-3.

The external fragmentation based on the number of buffers on the available list was in turn greater for the buddy method for the transition period than for the first-fit method. Since [1,2] when the algorithms were evaluated separately the opposite was found to be true, i.e., external fragmentation is more predominant in the first-fit than in the buddy method, an investigation of the adaptive strategy was needed. It was clear that the phenomenon was a result of attempting to make the two methods of allocation compatible and interchangeable. The analysis of the contributing factors is discussed in this section.

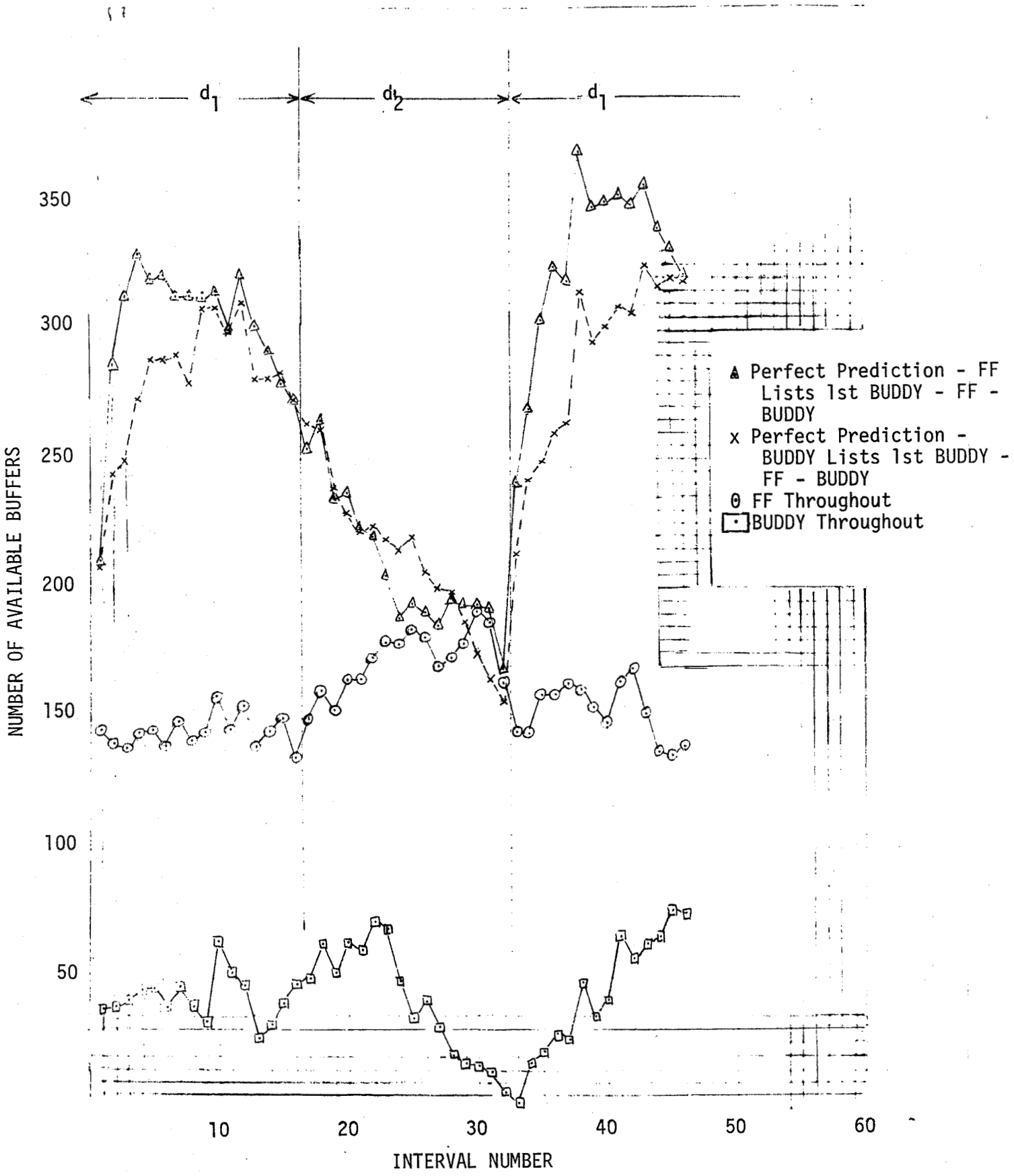


FIGURE III-3. COMPARISON OF NUMBER OF AVAILABLE BUFFERS ON LISTS.

3.2.1 Adaptive Strategy Model Simulated in [2] (Mod I)

A method for making a smooth transition from one allocation method to the other was worked out in [2]. One of the basic objectives which influenced the design of the transition mechanism implemented was to provide a rapid and complete transition. An attempt was made to limit the duration of interruption to the normal operation of either method and to keep algorithm memory requirements at a minimun, i.e., maintain only one algorithm in core.

In going from the buddy method to the first-fit method, the transition was effected immediately through an adjustment of available buffer list pointers. The space required by the buddy algorithm was then released. The duration of total changeover from the first-fit to the buddy method was also limited as far as possible. In this case, all allocations possible were made from the first-fit lists using the first-fit method until they were exhausted prior to allocation from the buddy lists. During this time, no buffers were returned to the first-fit lists but were returned to the buddy available lists. After a relatively short period of time, the first-fit lists were exhausted.

In the process of releasing buffers which were allocated using the first-fit method, buffers were checked for valid size, i.e., a power of two. If the sizes were not a power of two, a released buffer was split as required to insure a valid size and start address. Since in the first-fit allocation process buffers may be any multiple of four, it is clear that more buffers are allocated in the first-fit method which are not a power of two than are. In turn, the buddy release process creates many more available buffers than would be present if release were made using the first-fit method. The number of small buffers is

increased significantly and external fragmentation is unavoidably introduced.

An increase in small buffers is also introduced by the decision to make all allocations from the first-fit lists until they are exhausted. The small average request size is the basis for introducing the buddy method. In the process of exhausting the first-fit lists, all buffers on the first-fit list regardless of size are being used to satisfy the small buffer requests. This reduces the number of available large buffers unnecessarily. At the same time in the release process, small buffers are being created and returned to the buddy lists. These could be used to satisfy small buffer requests instead of splitting larger buffers from the first-fit list.

3.2.2 Modified Adaptive Strategy Model (Mod II)

As a result of the above observations, an alternative strategy for allocation during the transition period was modeled. In this model the buddy lists are checked and allocation are made from the buddy lists if possible. If this is not possible, the first-fit lists are checked and the allocation is made or the request is queued. Simulation runs showed that the number of available buffers was reduced over the original strategy, as seen in Figure III-3. When the memory pool was limited, queues were formed earlier using this strategy than the original one. See Figure III-4. This is caused since the internal fragmentation of the buddy method is significantly greater than that of the first-fit method. Allocations are being made using the buddy method primarily which does not make economical use of the limited storage available.

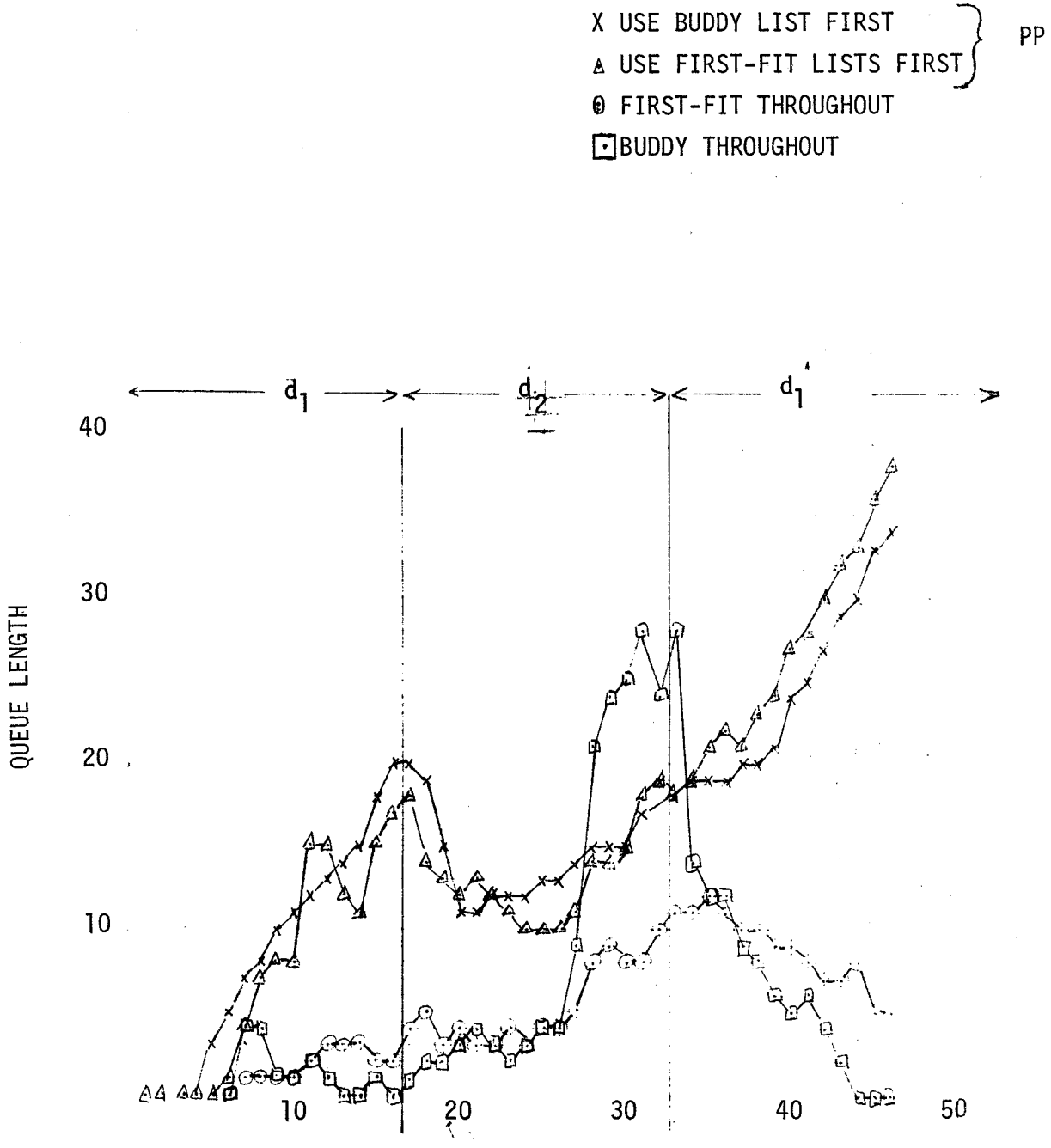


FIGURE III-4. QUEUE FORMATION IN FIRST-FIT, BUDDY, AND ADAPTIVE MOD WITH PERFECT PREDICTION OF REQUEST DISTRIBUTION IN USE.

Another serious consequence of this strategy is that the first-fit lists are not exhausted rapidly and the first-fit algorithm must reside in core indefinitely. In some simulation runs, the first-fit available lists were not exhausted throughout the period during which the buddy method was being employed.

3.2.3 Simulation Results of Mod I and Mod II

It was concluded from the simulation results of these two models that some reduction in the number of small buffers during the transition phase was possible using the modified model, Mod II. See Figure III-3. However it appears that this reduction is not significant based on the probability of queueing requests shown in Table III-3. There it is shown that if in the perfect prediction mode, the probability of queueing for Mod I is .025 and for Mod II is .022; or if using the predictor, the probability of queueing for Mod I is .007 and for Mod II is .005. In neither case is the difference signficant. In view of the above results and the fact that Mod II requires both allocation algorithms to remain in core indefinitely, Mod I would be preferable if an adaptive strategy were implemented.

3.3 Evaluation of Allocation Performance

The primary objective of the research discussed in this report was to explore the possibility of determining and implementing satisfactory criteria to control the use of alternative methods of the adaptive strategy. As the study progressed it became clear that a very basic problem should be explored. In order to compare the performance of the allocation process, we must be able to measure performance. The importance of a valid measure of allocation performance can not be overemphasized.

Memory utilization was selected as the basis for comparing alternative allocation strategies since algorithm execution times could not be obtained readily from the simulation outputs. The question to be answered is then 'What is a satisfactory measure of memory utilization?'. Several alternative measures have been discussed in the literature [4]. Among these are memory compaction and memory fragmentation.

Memory compaction is defined as the percent of the memory pool allocated when apparent memory overflow occurs. This may be viewed as a direct measure of memory utilization. This measure taken alone was considered inadequate since it is possible that one allocation strategy may provide a high degree of compaction but exhaust the memory pool more quickly than another. This introduces another element, internal fragmentation, which should be included in measuring allocation performance. For this reason attention was turned to finding a measure of memory fragmentation.

Memory fragmentation is a term used to define the amount of unusable memory. It is essentially an indirect measure of memory utilization since it measures the unusable memory as opposed to the used memory. Considered properly, it includes both internal and external loss. Initially, these two types of memory loss were analyzed independently. The results of this analysis showed that a satisfactory measure of total memory fragmentation could not be defined. The problem lay in the fact that internal loss and external loss are different and must be measured in different terms which could not be combined readily to obtain a resultant value of total memory loss. It was clear that a measure of total memory loss was still needed.

Approaching the problem of total memory fragmentation from another

viewpoint, the basic question is 'What is the significance of memory loss?'. If memory is unlimited, there is none. However, if memory is limited, one would like to use that allocation method where the probability of overflow is zero, or the smallest possible. This is the same as finding the probability that buffer requests will be queued. Since this could be measured from the simulation outputs and since this was a measure of total fragmentation regardless of the relative importance of internal or external fragmentation in the particular allocation method in use, the probability that buffer requests would be unhonored was the measure of allocation performance adopted. The following sections describe in detail the analysis which was performed and the results obtained.

3.3.1 Memory Fragmentation

The problem of memory fragmentation is two-fold. As discussed in Section 2 of this report, two distinct types of memory fragmentation are present - internal and external. It is clear that internal fragmentation is the predominant type of memory loss in the buddy method and external fragmentation is the predominant type of memory loss in the first-fit method. In the adaptive method which makes use of both the buddy and the first-fit method, the total fragmentation is the resultant of the two types. If the two types of fragmentation could be measured independently, then the sum of the results would be representative of the total fragmentation. This measure could then be used to compare the performance of the alternative methods of allocation.

The total fragmentation was obtained by adding the internal and external fragmentation. Plots were made of these three values. As

observed earlier, the plots gave an indication of the type of fragmentation which was characteristic of each method of allocation. In the buddy method internal fragmentation contributed most to the total fragmentation and in the first-fit external fragmentation was predominant. See Figure III-5.

This measure of total fragmentation was inadequate as a basis for comparing the allocation methods since external fragmentation does not realistically reflect the unusable memory or actual memory loss due to the breaking up of the memory pool into small buffers. Essentially, external fragmentation of memory is a matter of concern only when a request is made for a buffer which cannot be honored while at the same time the total amount of available storage exceeds this request. The number of small buffers available is of no consequence as a measure of external fragmentation if it is found that there is always an available buffer equal to or greater than the size requested.

From the above it is clear that external fragmentation is significant only for buffers of size x where there is a probability that $x > \max\{x_1\}$, where x_i are the sizes of the available buffers. From this it was concluded that the two important elements to be considered were the probability that there would be a request for a buffer of size x and the probability that an available buffer of size x or greater would not be available. The effect of fragmentation was then defined in terms of the probability that a request for size 2^i would be queued as follows:

$$p(q|2^i) = \Phi(2^i) \cdot \prod_{k=i}^{n} p(2^k \text{ list empty}) \qquad (3\text{-}5)$$

where $\Phi(2^i)$ is the probability of a request for a buffer of size 2^i,

$p(2^k \text{ list empty})$ is the probability that the 2^k list is empty,

$p(q|2^i)$ is the probability that, given a request for 2^i, the request

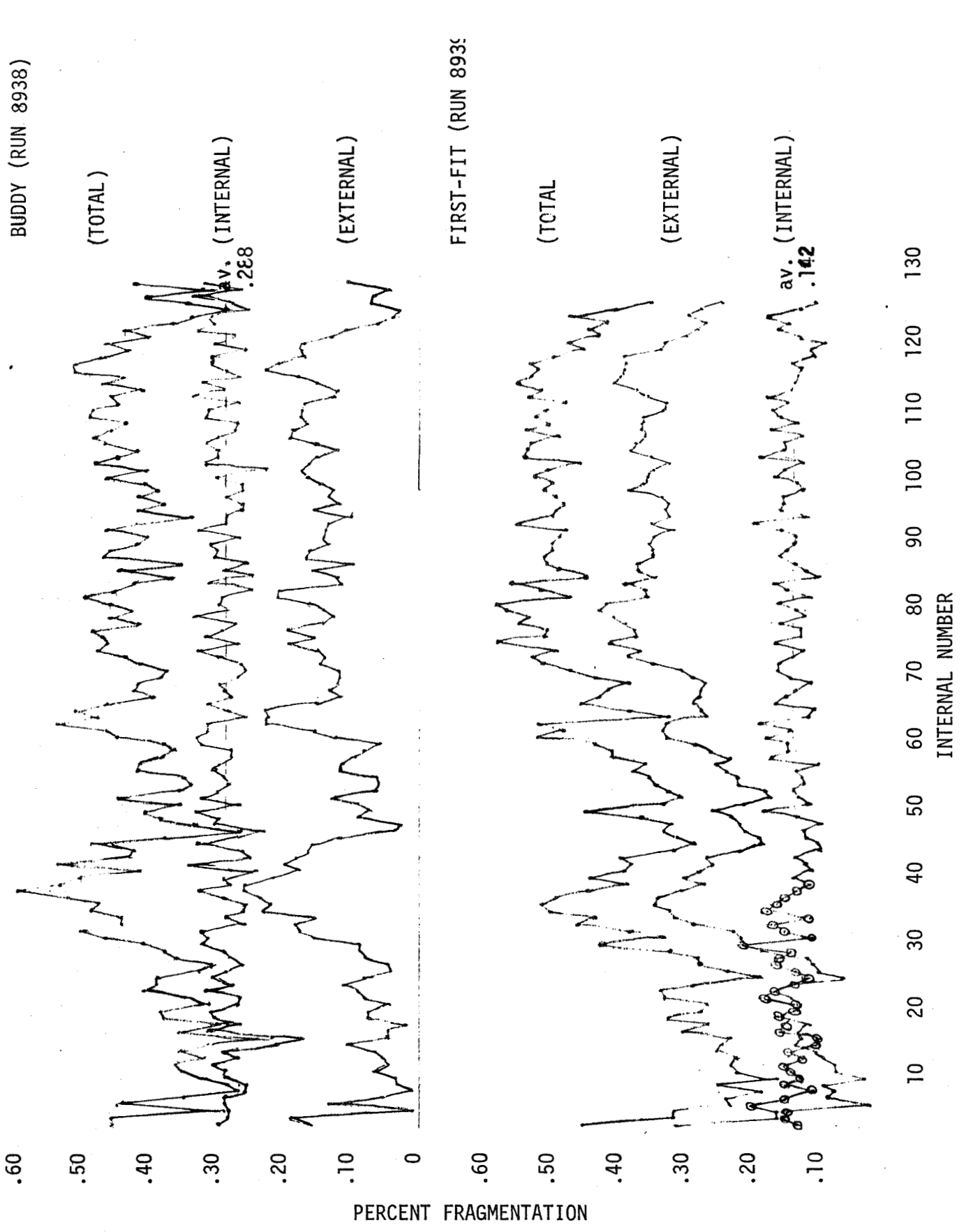


FIGURE III-5. MEMORY FRAGMENTATION BASED ON INTERNAL AND EXTERNAL FRAGMENTATION

will be queued.

If using the first-fit allocation method, the calculation for determining when a request will be queued requires additional analysis. This is required since, unlike in the buddy method, a list may not be empty but may contain buffers which are not a power of two. This allows for the situation where a buffer less than 2^i exists on the 2^i list and a request is made for a buffer greater than the available buffer. When this occurs, the request is queued even though the list, 2^i, is not empty. In order to obtain the probability of queueing for the first-fit allocation, the following modification was made:

$$(3\text{-}6) \qquad p(q|2^i) = \Phi(2^i) \cdot \prod_{k=i+1}^{n} p(2^k=0) \cdot \cdot [p(2^i=0) + \sum p(\text{req} > \max\{2^i\})]$$

where $p(\text{req} > \max\{2^i\})$ is the probability that a request is made for a buffer which is greater than the maximum size buffer on list in an interval.

Finally, for either the first-fit or the buddy method, the probability that any request will be queued and not honored given all requests is given by

$$(3\text{-}7) \qquad p(q) = \sum_{i=2}^{n} p(q|2^i).$$

The outputs from the simulation of each model were evaluated in terms of the above interpretation of memory fragmentation. The particularly important aspect of this approach is that fragmentation need not be treated as two distinct types - internal and external. This interpretation of fragmentation avoids the problems of measuring and comparing dissimilar entities in an effort to arrive at a single measure

based on their contributory components. The results obtained reflect the effect of total fragmentation, regardless of the type.

In the simulation the probability of requests for buffers of size 2^i, i=2,...,8, was obtained directly from the request distributions used to generate the requests. The probability that a list of 2^i size buffers would be empty was then obtained by plotting the frequency distribution of the number of buffers on each available buffer list at the end of each processing interval. Additional simulation runs were made in order to insure that steady state in the allocation process had been reached and to permit a significant period of time over which the frequency distribution was obtained. The percentage of the time that each list was empty was then used as the probability that a given list would be empty. If any list were never found to be empty, then the probability that this list would be empty was set to zero along with all lists which contained smaller buffers.

3.3.2 Scope of Simulation

In this simulation study two distributions, d_1 and d_2, were used to generate the buffer requests. The average buffer request sizes were $\overline{d}_1 = 15.18$ and $\overline{d}_2 = 20.25$ words respectively. See Figure III-1. The simulation runs were set up so that the outputs could be used to compare the allocation performance of the first-fit method, the buddy method throughout, and the two models of the adaptive strategy discussed in Section 3.2. For both of the adaptive models, the allocation algorithm in use was controlled in two ways. The first, a perfect prediction of request distribution change, was controlled as a function of the distribution in use. In the second, the decision to use a particular

algorithm was based on monitoring the average request size, the change in the average request size, and the formation of queues of unhonored requests. This latter method is that method described in Section 3.1. Table III-2 provides a chart of the simulation runs performed.

3.3.3 Results of Simulation

In using distribution d_1 with an average request size of 15.18 words, it was found that neither the buddy nor the first-fit method of allocation ever exhausted the largest size buffer list, 2^9. The probability that a queue of unhonored requests would exist was zero. For this distribution the buddy method was found to be slightly better than the first-fit method in terms of memory utilization. This was based on the fact that throughout the use of the buddy method, there were always two or more buffers of size 2^9 available, while the first-fit method reduced the 2^9 list to just one buffer.

This is a rather interesting result in view of the fact that it was found in [2] that based on internal fragmentation alone whenever the average allocated size in the buddy method is greater than four times the average overhead of the first-fit method, more memory will be used by the buddy method than by the first-fit method. Here the average overhead of the first-fit method is 2.5 words and 4x2.5 = 9 would be the point at which the first-fit method would be preferable. However, this run shows that with an average allocation size of 21.24, the buddy method is still slightly more economical of memory available than the first-fit method. This indicates that external fragmentation is significant and must be considered if a meaningful comparison of the two methods is to be made.

	Buddy	First-Fit	Perfect Prediction FF-B Mod 1	Perfect Prediction FF-B Mod 2	Perfect Prediction B-FF Mod 2	Predicted FF-B Mod 1	Predicted FF-B Mod 2
d_1	x	x					
d_2	x	x					
$d_1 \rightarrow d_2$	x	x			x		
$d_2 \rightarrow d_1$	x	x	x	x		x	x

TABLE III-2. SIMULATION RUNS PERFORMED. (NOTE MOD 1 AND MOD 2 ARE ADAPTIVE STRATEGY MODELS WHERE 'USE FIRST-FIT LISTS FIRST' OR 'USE BUDDY LISTS FIRST' RESPECTIVELY AS DESCRIBED IN SECTION 3.2.)

Interval #	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9
39	48	17	40	31	8	5	4	1
41	48	23	44	31	7	4	5	1
43	52	29	39	27	14	3	2	1(512)*
45	56	33	35	25	11	4	2(188)	0
47	58	25	36	27	3	2(68)	0	0
49	69	23	45	40	4	3(128)	0	0
51	72	26	48	47	4	1(84)	0	0
53	67	31	47	44	10(56)	0	0	0
55	53	25	50	49	23(64)	0	0	0
57	57	18	47	46	28	6(100)	0	0
59	60	21	42	45	18	5(96)	0	0
61	50	22	37	45	22	12(116)	0	0
63	56	22	38	41	23	11	1(136)	0
65	66	19	41	38	24	10(96)	0	0
67	64	18	37	37	26	12(100)	0	0
69	74	21	40	26	17	10	1(156)	0
71	69	23	34	25	19	6(96)	0	0
73	63	15	33	37	20	7(112)	0	0
75	68	15	41	39	11	4	1(192)	0
77	86	26	43	36	5(52)	0	0	0

TABLE III-3. NUMBER OF AVAILABLE BUFFERS ON EACH LIST AT THE END OF PROCESSING INTERVAL USING DISTRIBUTION d_2 WITH AVERAGE BUFFER REQUEST SIZE d_2 = 20.25 AND THE FIRST-FIT ALLOCATION METHOD.

(*NOTE. THE NUMBER IN PARENTHESIS IS THE LARGEST BUFFER AVAILABLE ON ANY LIST.)

Interval #	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9
39	0	7	12	2	8	4	0	0
41	2	11	10	0	3	1	3	0
43	3	9	6	1	5	2	1	0
45	10	3	3	6	4	0	0	0
47	1	1	1	0	0	0	0	0
49	0	4	6	0	0	0	0	0
51	8	1	1	0	0	0	0	0
53	9	7	0	0	0	0	0	0
55	6	5	0	0	0	0	0	0
57	3	7	2	1	0	0	0	0
59	6	9	10	9	0	0	0	0
61	5	6	2	0	0	0	0	0
63	5	13	4	4	0	0	0	0
65	1	12	14	15	0	0	0	0
67	1	2	15	14	0	0	0	0
69	3	4	0	2	0	0	0	0
71	3	2	1	6	0	0	0	0
73	2	2	1	10	6	0	0	0
75	3	12	10	4	4	0	0	0
77	6	11	15	11	9	0	0	0

TABLE III-4. NUMBER OF AVAILABLE BUFFERS ON EACH LIST AT END OF PROCESSING INTERVAL USING DISTRIBUTION a_2 WITH AVERAGE BUFFER REQUEST SIZE $\bar{d}_2$ = 20.25 WITH THE BUDDY ALLOCATION METHOD.

(NOTE: EVERY BUFFER ON LIST IS A POWER OF TWO WITH BUDDY ALLOCATION METHOD.)

	Buddy	First-Fit
$p(q \mid x < 2^2)$	.0	.0
$p(q \mid 2^2 \leq x < 2^3)$	.0	.0
$p(q \mid 2^3 \leq x < 2^4)$	.0065	.0
$p(q \mid 2^4 \leq x < 2^5)$	.1037	.0
$p(q \mid 2^5 \leq x < 2^6)$	.0302	.0002
$p(q \mid 2^6 \leq x < 2^7)$	.0144	.0041
$p(q \mid 2^7 \leq x < 2^8)$	.0090	.0067
$\sum p(q \mid 2^i)$	.1638	.0110

TABLE III-5. PROBABILITY OF QUEUEING GIVEN BUFFER REQUESTS FROM DISTRIBUTION d^2 WITH AVERAGE BUFFER REQUEST SIZE $\bar{d}^2 = 20.25$. (SEE WORKSHEET 1 AND 2 FOR CALCULATIONS USING EQUATIONS 3-5, 3-6, and 3-7).

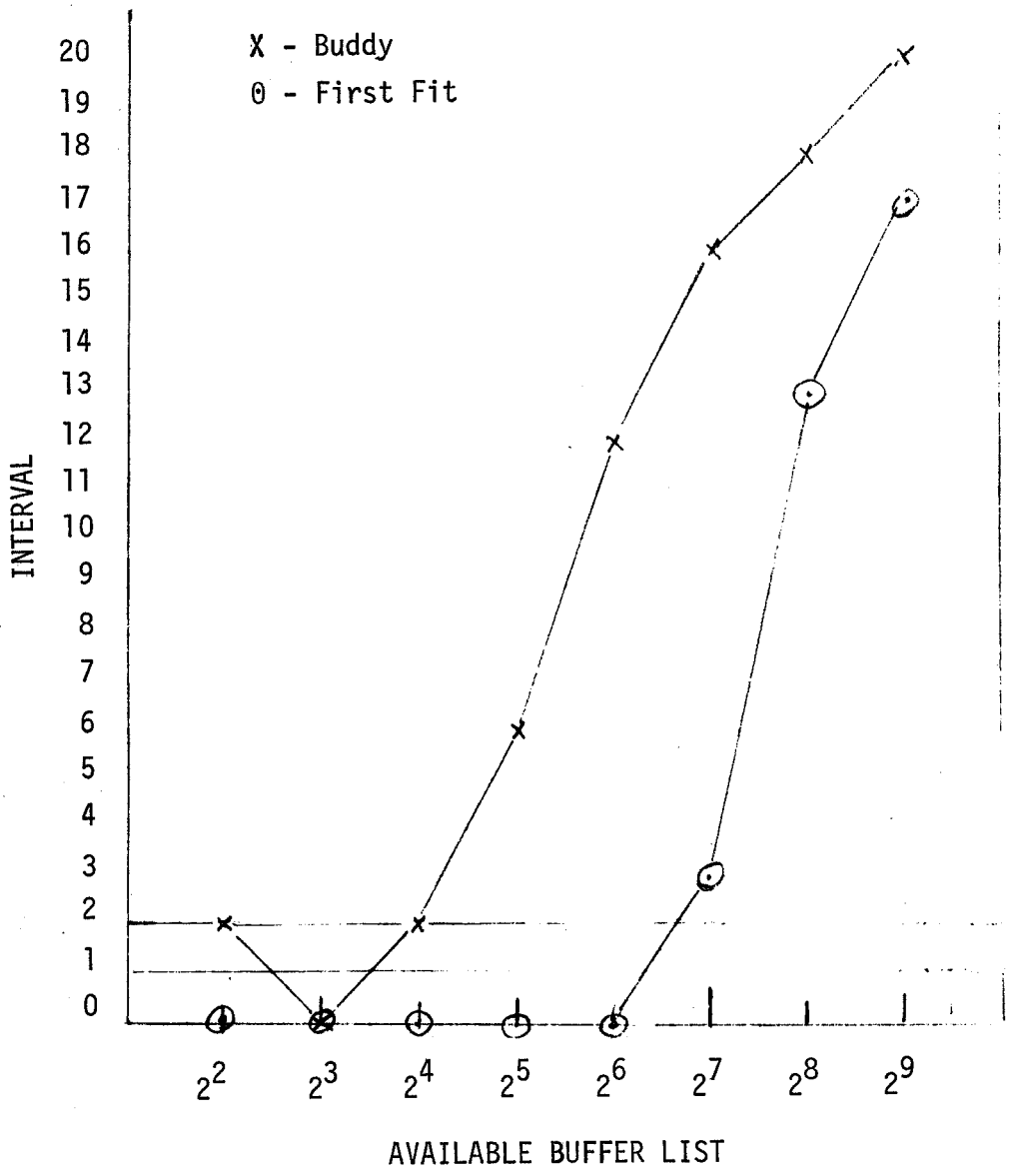


FIGURE III-6. FREQUENCY WITH WHICH A LIST IS EMPTY. USING DISTRIBUTION d_2 WITH AVERAGE BUFFER REQUEST SIZE $d_2 = 20.25$.

$p(2^9 = 0) = 1.0$ $\quad\quad$ $\Phi(2^9) = 0$

$p(2^8 = 0) = .9$ $\quad\quad$ $\Phi(2^8) = .01$

$p(2^7 = 0) = .8$ $\quad\quad$ $\Phi(2^7) = .02$

$p(2^6 = 0) = .6$ $\quad\quad$ $\Phi(2^6) = .07$

$p(2^5 = 0) = .3$ $\quad\quad$ $\Phi(2^5) = .40$

$p(2^4 = 0) = .1$ $\quad\quad$ $\Phi(2^4) = .25$

$p(2^3 = 0) = 0.0$

$$p(q|2^8) = \Phi(2^8) \cdot \prod_{k=8}^{9} p(2^k = 0) = (.01)(1.0)(.9) = .009$$

$$p(q|2^7) = \Phi(2^7) \cdot \prod_{k=7}^{9} p(2^k = 0) = (.02)(1.0)(.9)(.8) = .0144$$

$$p(q|2^6) = \Phi(2^6) \cdot \prod_{k=6}^{9} p(2^k = 0) = (.07(1.0)(.9)(.8)(.6) = .0302$$

$$p(q|2^5) = \Phi(2^5) \cdot \prod_{k=5}^{9} p(2^k = 0) = (.40)(1.0)(.9)(.8)(.6)(.3) = .1037$$

$$p(q|2^4) = \Phi(2^4) \cdot \prod_{k=4}^{9} p(2^k = 0) = (.25)(1.0)(.9)(.8)(.6)(.3)(.1) = .0065$$

$$p(q) = \sum_{i=2}^{9} p(q|2^i) = .0090 + .0144 + .0302 + .1037 + .0065$$

$$= 0.1648$$

WORKSHEET 1. CALCULATION OF PROBABILITY OF QUEUEING A REQUEST USING BUDDY METHOD WITH DISTRIBUTION d_2. DATA TAKEN FROM TABLE III-4 AND FIGURE III-1. EQUATIONS (3-5) AND (3-7) USED.

$p(2^9 = 0) = .85$ $\quad\quad$ $\Phi(2^9) = 0$

$p(2^8 = 0) = .65$ $\quad\quad$ $\Phi(2^8) = .01$

$p(2^7 = 0) = .15$ $\quad\quad$ $\Phi(2^7) = .02$

$\Phi(2^6) = .07$

$$p(q|2^7 \leq x < 2^8) = \Phi(2^8) \cdot \prod_{k=9} p(_{2=0}^{k}) \cdot [p(2^6=0) + \sum p(\text{req} > \max\{2^6\})]$$

$$= (.01)(.85) \cdot [(.65) + \frac{68+120+100+64}{128} \cdot \frac{1}{20}]$$

$$= (.01)(.85)(.7875) = .0067$$

$$p(q|2^6 \leq x < 2^7) = (.02)(.85)(.65)[.15 + \frac{1}{20} \quad \frac{60+44+2x28+3x32+12+16}{64} \quad]$$

$$= (.02)(.85)(.65)(.3719) = .0041$$

$$p(q|2^5 \leq x < 2^6) = (.07)(.85)(.65)(.15)[0 + (\frac{8+12}{32}) \cdot \frac{1}{20}]$$

$$= (.07)(.85)(.65)(.15)(.0313) = .0002$$

$$p(q) = \sum_{i=2}^{9} p(q|2^{i-1} \_ x \quad 2^i) = .0067 + .0041 + .0002 = \underline{\underline{.0110}}$$

WORKSHEET 2. CALCULATION OF PROBABILITY OF QUEUEING A REQUEST USING FIRST-FIT METHOD WITH DISTRIBUTION d_2. DATA TAKEN FROM TABLE III-3 AND FIGURE III-1, EQUATIONS (3-6) AND (3-7) USED.

In distribution d_2 with an average buffer request size of 20.25, requests were queued using either the first-fit or the buddy method. For this distribution the allocation performance of the two methods was compared using the probability of queueing method discussed above. Here the probability that requests will be queued was found to be .011 for the first-fit method and .164 for the buddy method. Tables III-3 and III-4 give the frequency distribution of the number of blocks on the available lists for the first-fit and buddy methods respectively. Figure III-6 shows the frequency with which each list is empty using either the buddy or the first-fit method. Table III-5 gives the probability that a request for a buffer of 2^i, i=2,...,8, will be queued for the two methods. The total probability of queueing is the result of summing these probabilities. A direct comparison of the performance of the alternative allocation strategies is possible from the values given in this table.

Similar tables and calculations were performed for each simulation run. The results, the total probability of queueing for each run, are given in Table III-6. It should be pointed out that in all runs data from processing intervals 39 to 77 were used. This permits the use of data obtained after steady state conditions have been established and provides a common processing period for the comparison of the allocation methods.

3.4 Summary of Conclusions

The simulation results show:

(1) The predictor method of controlling the adaptive method is feasible and provides adequate control in view of the distributions used

	Buddy	First-Fit	Perfect Prediction FF-B Mod I	Perfect Prediction FF-B Mod II	Perfect Prediction B-FF Mod II	Predictor FF-B Mod I	Predictor FF-B Mod II
d_1	.00	.00					
d_2	.160	.011					
$d_1 \rightarrow d_2$	.040	.002			.016		
$d_2 \rightarrow d_1$	.038	.008	.025	.022		.007	.005

TABLE III-6. PROBABILITY OF QUEUEING RESULTS BASED ON SIMULATION RUNS (NOTE MOD I AND MOD II ARE ADAPTIVE STRATEGY MODELS WHERE 'USE FIRST-FIT LISTS FIRST' OR 'USE BUDDY LIST FIRST' RESPECTIVELY AS DESCRIBED IN SECTION 3.2).

to generate the buffer requests.

(2) The result indicates that the control of the allocation algorithm in use should be based not only on the average buffer request size but also on the existence of queued requests.

(3) The attempt to improve allocation performance by reducing the numbers of small buffers placed on the available buffer lists during the transition from the first-fit to the buddy method did not produce significant improvement due to the basic incompatibility of the first-fit and buddy methods of allocation. If one decides to use the adaptive strategy, the model (Mod I), which if possible uses buffers from the first-fit lists prior to allocating from the buddy lists, should be implemented. This avoids the necessity of maintaining both algorithms, the first-fit and the buddy, in core indefinitely.

(4) A comparison of the performance of the allocation methods simulated is possible using the probability of queueing calculation proposed in this report.

(5) The attractive feature of using the probability of queueing as a basis for comparing allocation performance is that it reflects the effect of total fragmentation which includes both internal and external fragmentation. This avoids the problem of determining the relative importance of internal or external fragmentation individually in an effort to arrive at a measure of total fragmentation.

(6) Finally Table III-6 shows that for the request distribution used in the simulation the adaptive strategy is slightly better than the first-fit method and significantly better than the buddy method. The difference in the allocation performance of the first-fit method and the adaptive strategy is clearly not significant. Further in view

of the need for monitoring, predicting when to alternate allocation strategies, and the added complexity of the individual algorithms to make them compatible in the adaptive mode, one would be advised to implement the first-fit method.

(7) Simulation is an effective tool for studying the characteristics of alternate methods of handling computer operating system functions prior to modifying an existing system or including a proposed strategy in a proposed system design.

4. Bibliography

[1] Minker, J., Crooke, S., Yeh, J., "Analysis of Data Processing Systems", University of Maryland Technical Report No. 69-99, December 1969, p. 103.

[2] Crooke, S. An Adaptive Approach to the Allocation of Buffer Storage, M.S. Thesis in Computer Science, University of Maryland, College Park, Maryland, June 1970, p. 78.

[3] Markowitz, H. M., Hausner, B., Karr, H. W., Simscript: A Simulation Programming Language, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1963.

[4] Knuth, D. E., The Art of Computer Programming, Vol. 1, Fundamental Algorithms, Addison-Wesley, Menlo Park, California, 1968.

[5] Denning, P. J. "Virtual Memory", Dept of E. E. Computer Science Lab. Technical Report No. 81, January 1970, p. 87.

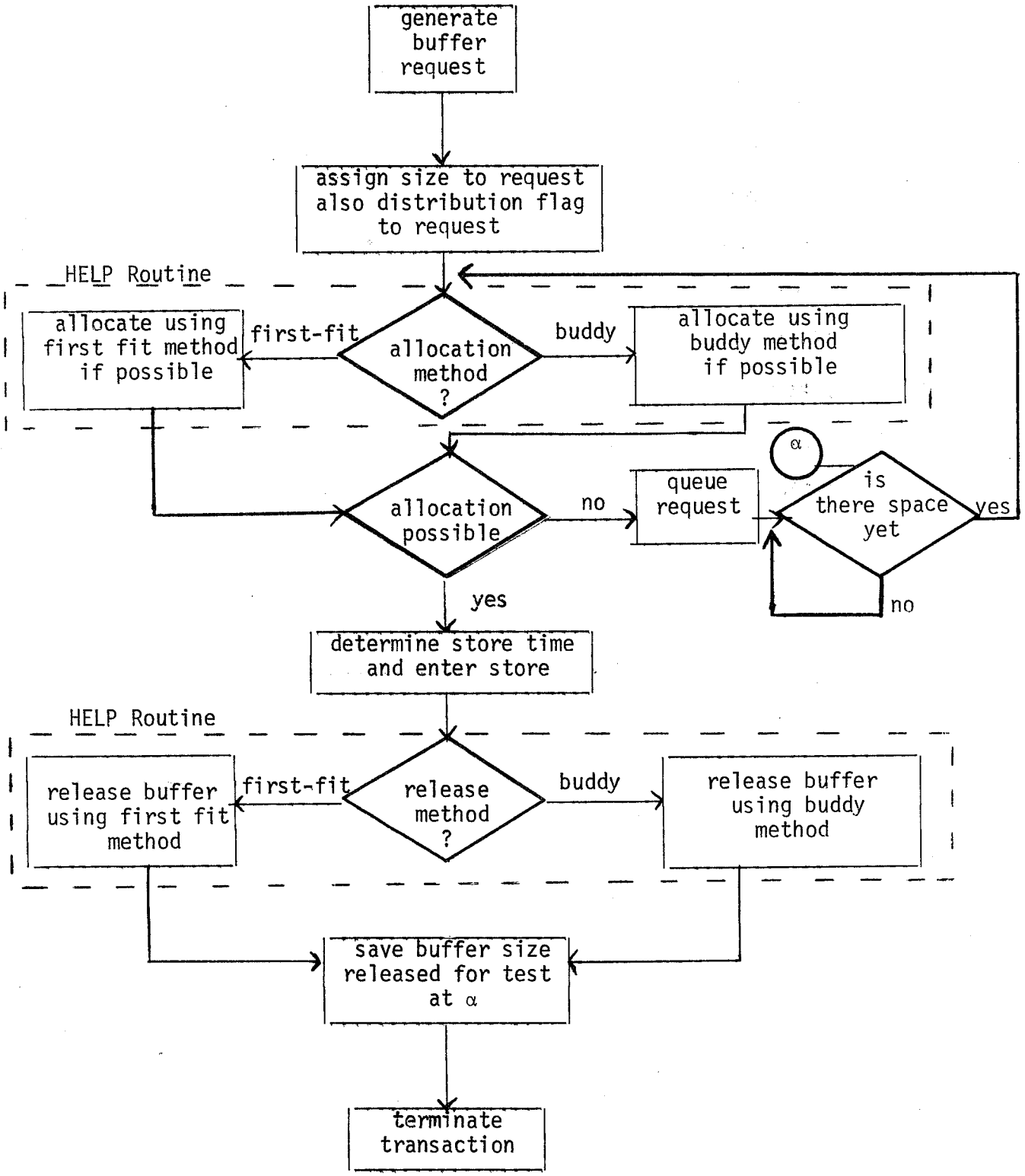
[6] Randell, B., Kuehner, C. J., Dynamic Storage Allocation Systems. CACM 11, 5 (May 1968) 297-305.

[7] General Purpose Systems Simulator II (GPSS) Reference Manual, Univac Data Processing Division Manual Number UP-4129.

Appendix

The purpose of this appendix is to present in some detail the logic and data structure of the allocation methods simulated. First a skeletal outline of the total model is given. This is followed by a detailed flow chart of the GPSS routine which serves as the basic control in the simulation. A flow chart of the HELP routine* gives the models used in simulating the first-fit, buddy, and adaptive allocation methods. Finally, the data structure employed to make the first fit and buddy allocation methods compatible is discussed. It is hoped that the discussion and detail given in this appendix will permit a better understanding of the simulation performed and the results which were obtained.

*Note: HELP is a Fortran routine called using a standard GPSS block type.



A-1. GENERAL OUTLINE OF SIMULATION MODEL

In GPSS, each transaction may have eight words, P1,...,P8, associated with it and may be used by the programmer to describe the transaction. In this simulation, the following use is made:

P1 - The actual buffer request size as obrained using request distribution FN1 or FN3. (See Figure III-1 for graph of these functions.)

P2 - The start location of the buffer allocated.

P3 - the exponent of the least power of two which is equal to or greater than the buffer request size in P1.

P4 - The actual buffer size allocated. If allocation is made using the first-fit method, P4 is the multiple of four which is equal to or greater than the request size contained in P1. If allocation is made using the buddy method, P4 is a power of two.

P5 - P8 unused.

Certain storage may be used for temporary storage and may be referenced by Xn where n is the cell number in the temporary storage area. In this program X1 through X6 are used as follows:

X1 - Indicates either a request for buffer allocation, or a request for buffer release upon entering the HELP routine. Upon return from the HELP routine, X1 contains the start location of the buffer allocated.

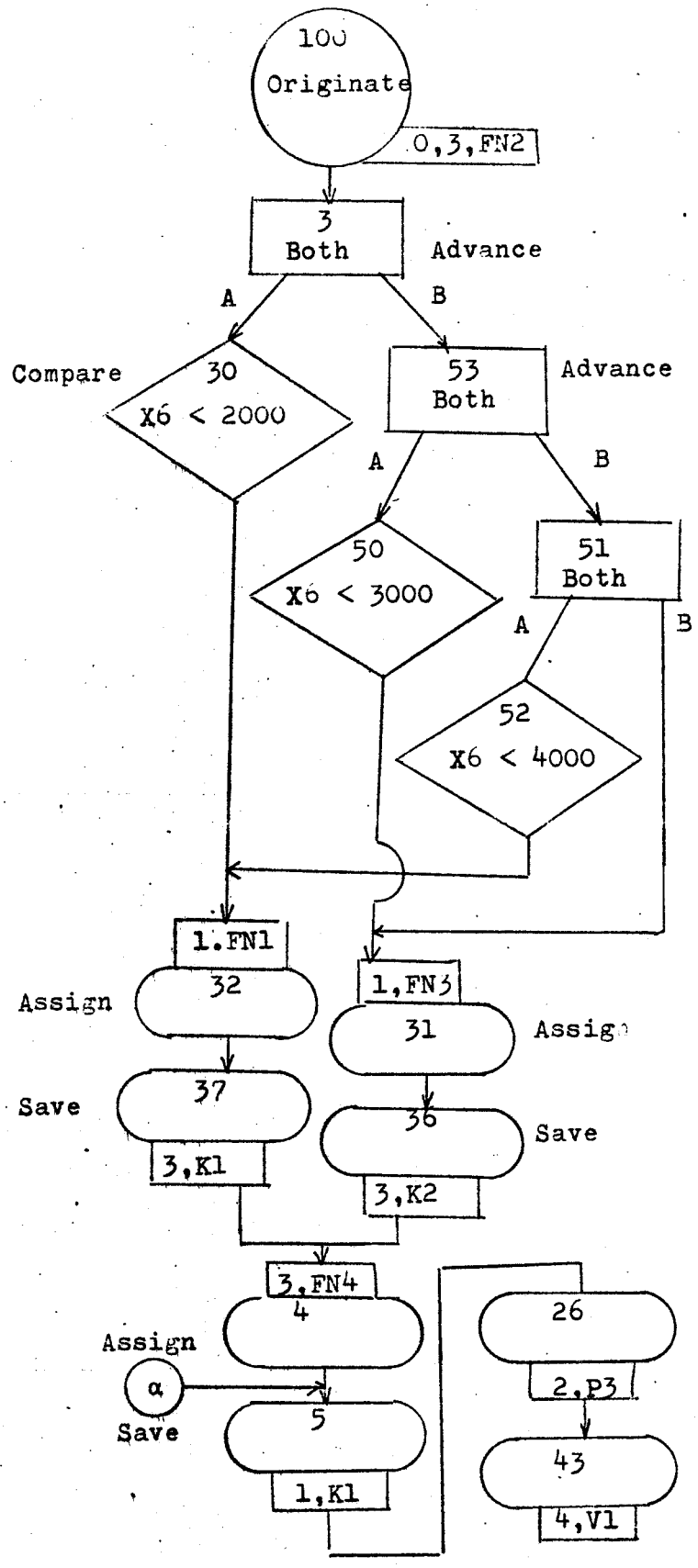
X2 - Contains the least power of two which is equal to or greater than the buffer requested. Upon exit from the HELP routine, X2 contains size of buffer allocated.

X3 - Indicates which allocation method should be used if control is to be a function of the request distribution in use.

X4 - Contains total number of requests currently queued.

X5 - Size of most recently released buffer.

X6 - Total number of releases.

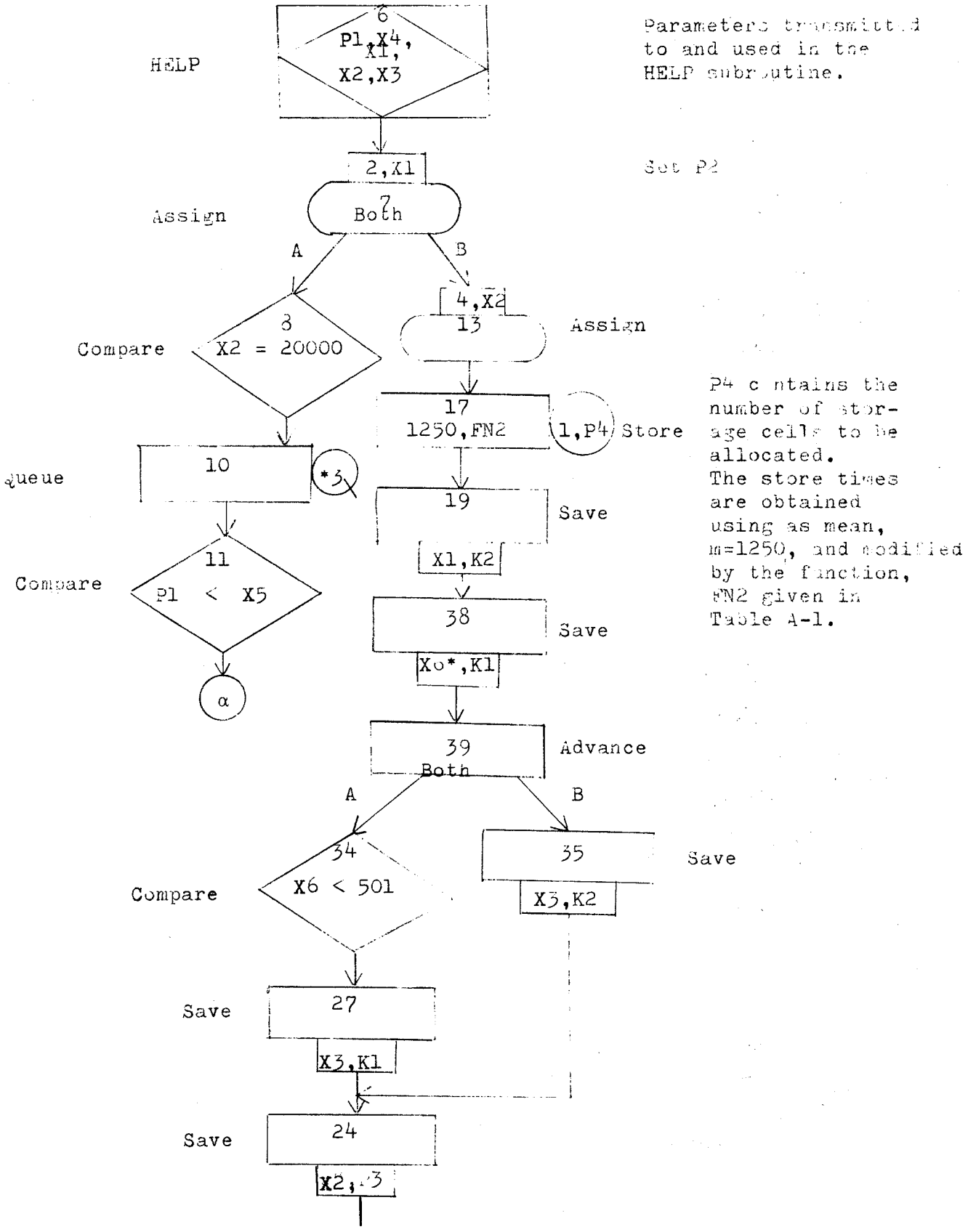


Transactions (buffer requests) are originated using a Poisson distribution. This distribution is obtained using the mean inter-ar ival time, m=3, and modified by the function FN2 as given in Table A-1.

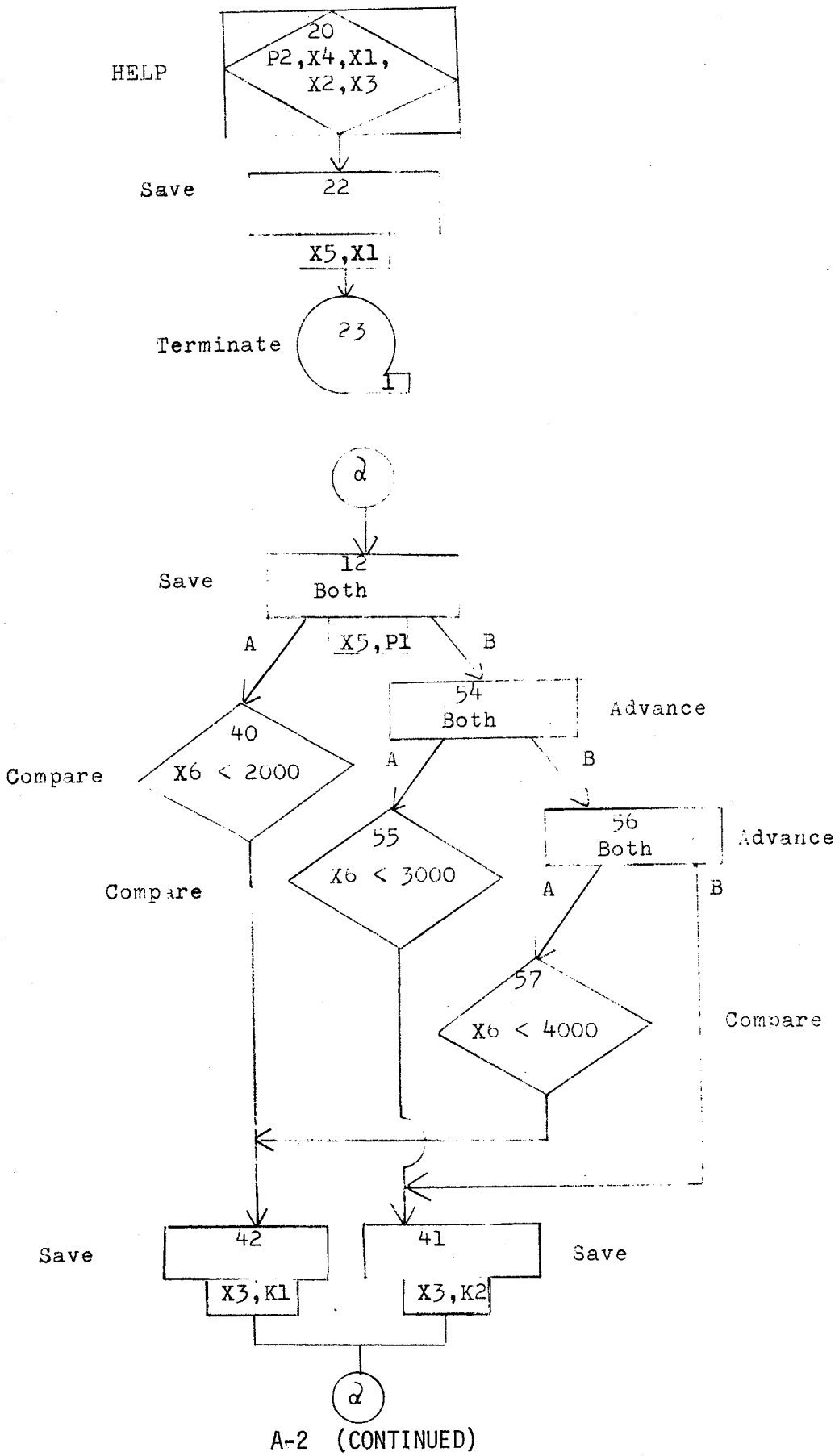
Determine request distribution to be used and X3 as a function of the value of X6 where X6 is the number of processed requests, that is, the number of released buffers.

Set P1 = the actual buffer request size obtained using request distribution FN1 or FN3.
Set X3 to indicate which allocation method should be used as a function of the request distribution used, FN1 or FN3.
Set P3 = the least power of two which is equal to or greater than the buffer request size in P1.
Set X1 = to indicate request for buffer allocation.
Set X2 = power of two found in P3.
Set X4 = total number of requests queued at the time current request processed.

A-2. GPSS CONTROL PROGRAM



A-2 (CONTINUED)



A-2 (CONTINUED)

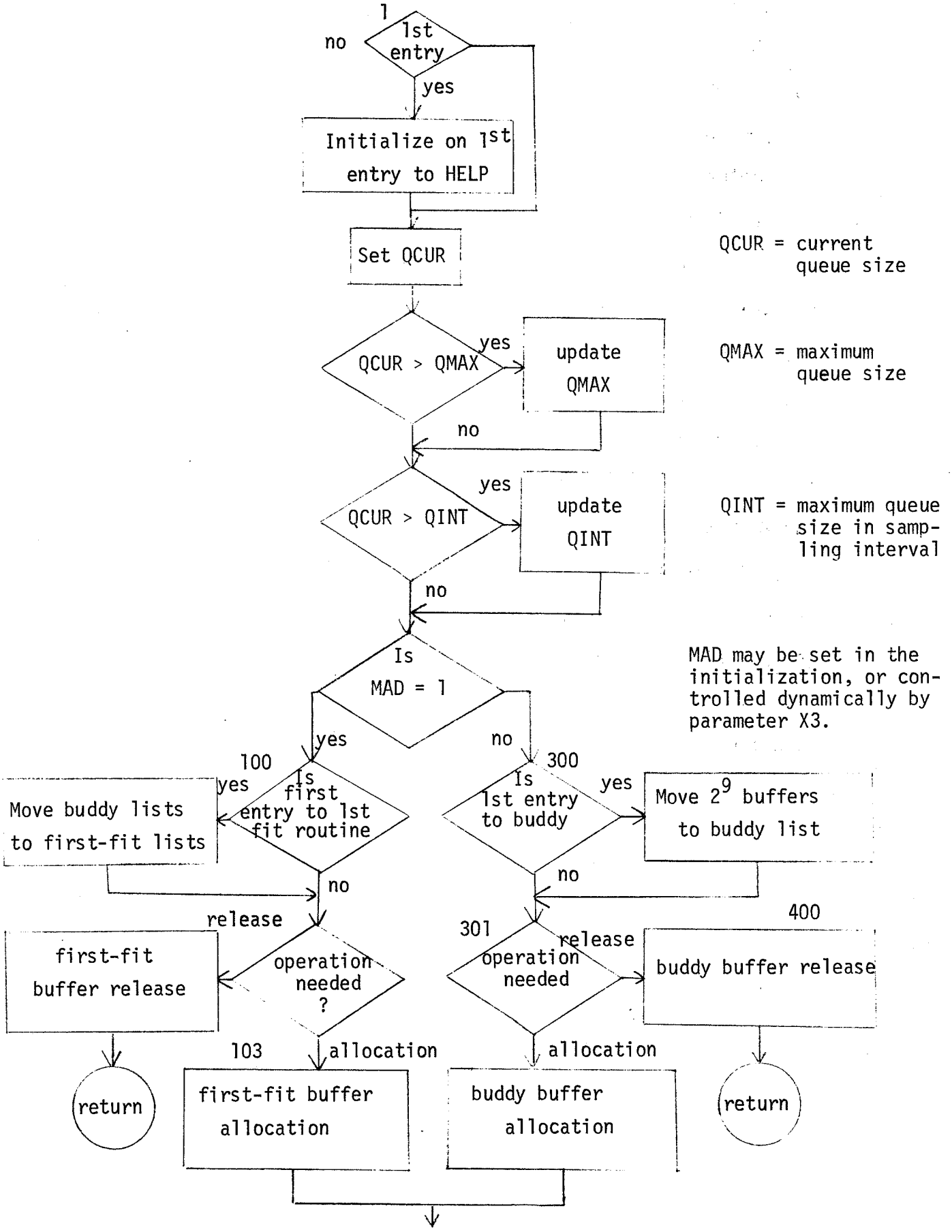
X	Y	X	Y
0	0	.9	2.3
.1	.104	.92	2.52
.2	.222	.94	2.81
.3	.355	.95	2.99
.4	.509	.96	3.2
.5	.690	.97	3.5
.6	.915	.98	3.9
.7	1.200	.99	4.6
.75	1.38	.995	5.3
.8	1.6	.998	6.2
.84	1.83	.999	.7
.88	2.12	.9997	.8

TABLE A-1. FUNCTION FN2 - EXPONENTIAL DISTRIBUTION

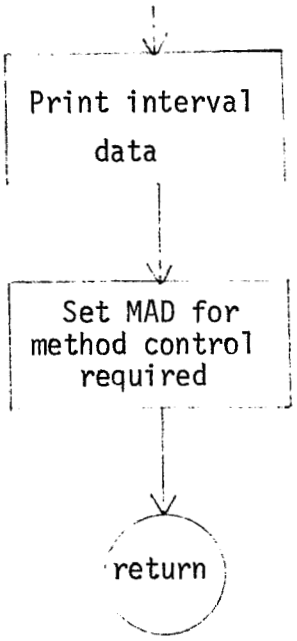
The HELP subroutine is written in Fortran. Its basic function is to maintain the memory map and data structure of the buffer pool. Two methods of buffer allocation are provided: the first-fit and the buddy method. The data structure maintained is such that the methods may be used interchangeably. Provision is made for interchanging the methods as a function of the request distribution in use. The request distribution may be determined either from a parameter passed through the calling sequence to HELP or by calculating the average request size and the observed change in the average request size.

The flow chart of the HELP routine outlines the models used for the first-fit allocation and release processes, the buddy allocation and release processes, the method of calculating the average request size and the change in request size, and the means provided for changing from one allocation method to the other as a function of the request distribution.

Finally, the buffer pool data structure which permits the use of either allocation method is described.

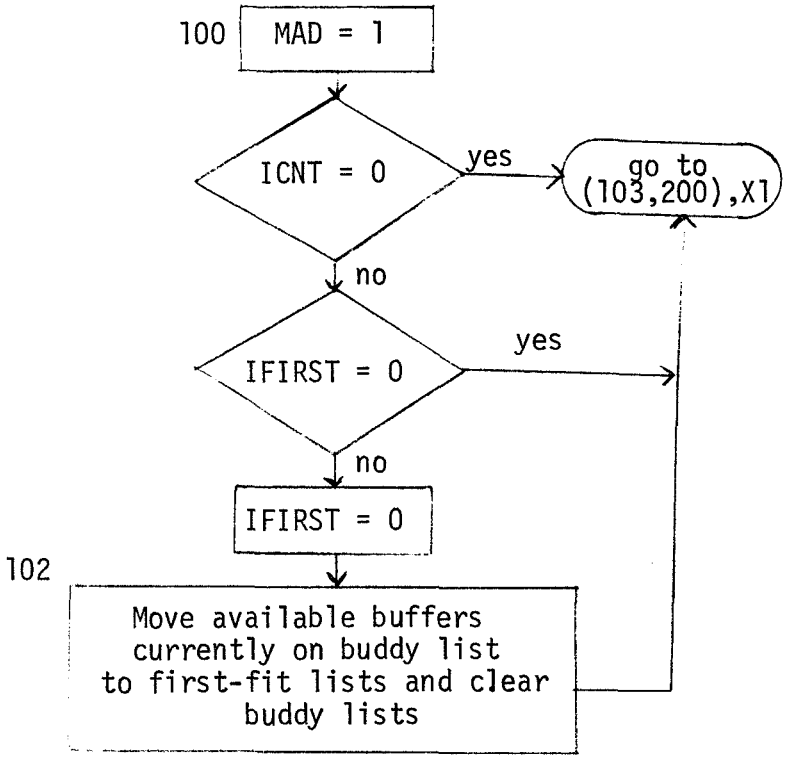


A-3. GENERAL OUTLINE OF HELP ROUTINE



A-3 (CONTINUED)

First-Fit Routines



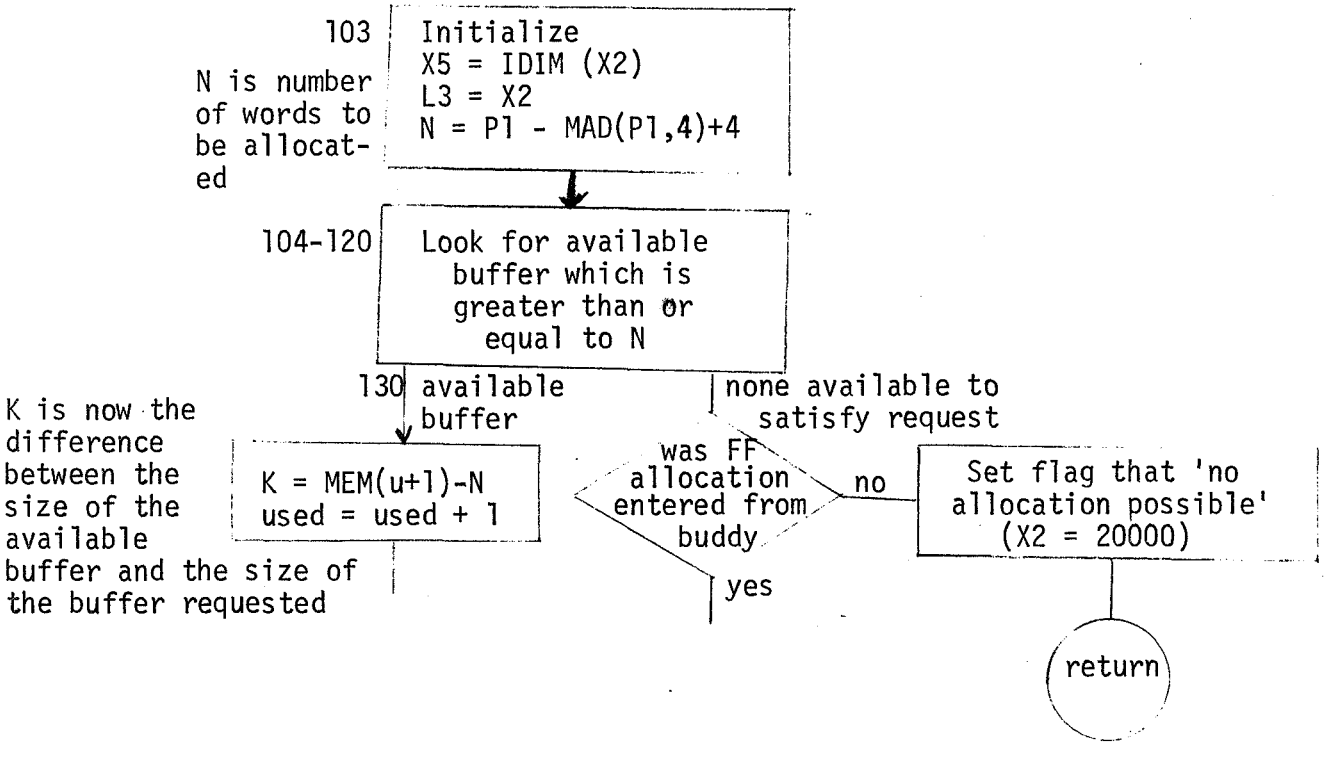
X1 = 1 for buffer allocation
= 2 for buffer release

MEM(_) is pointer to next buffer on list

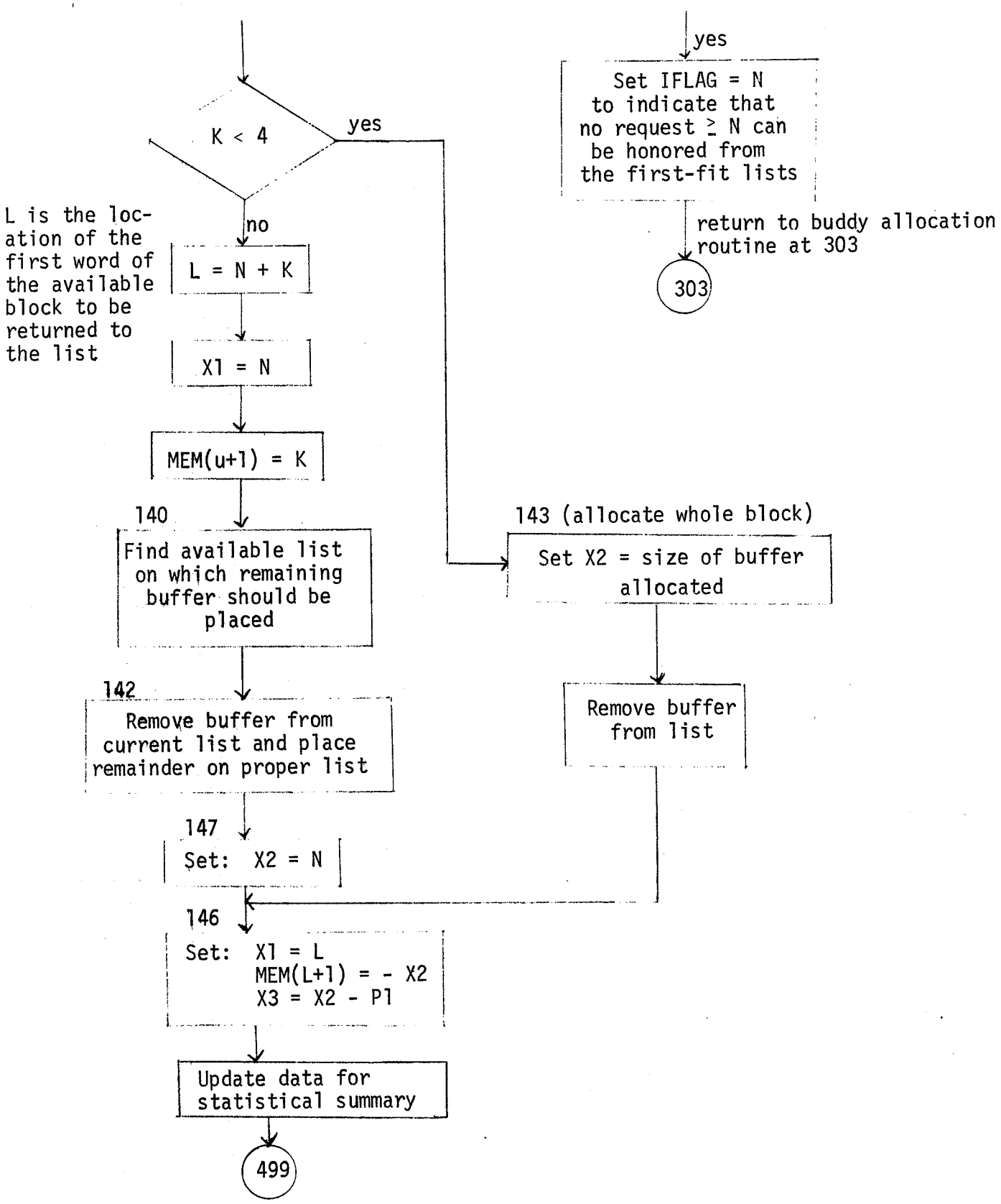
MEM(_+1) is the size of the buffer. If the buffer is allocated, the sign is minus.

MEM(_+2)is the list on which the buffer is found.

First-Fit Allocation

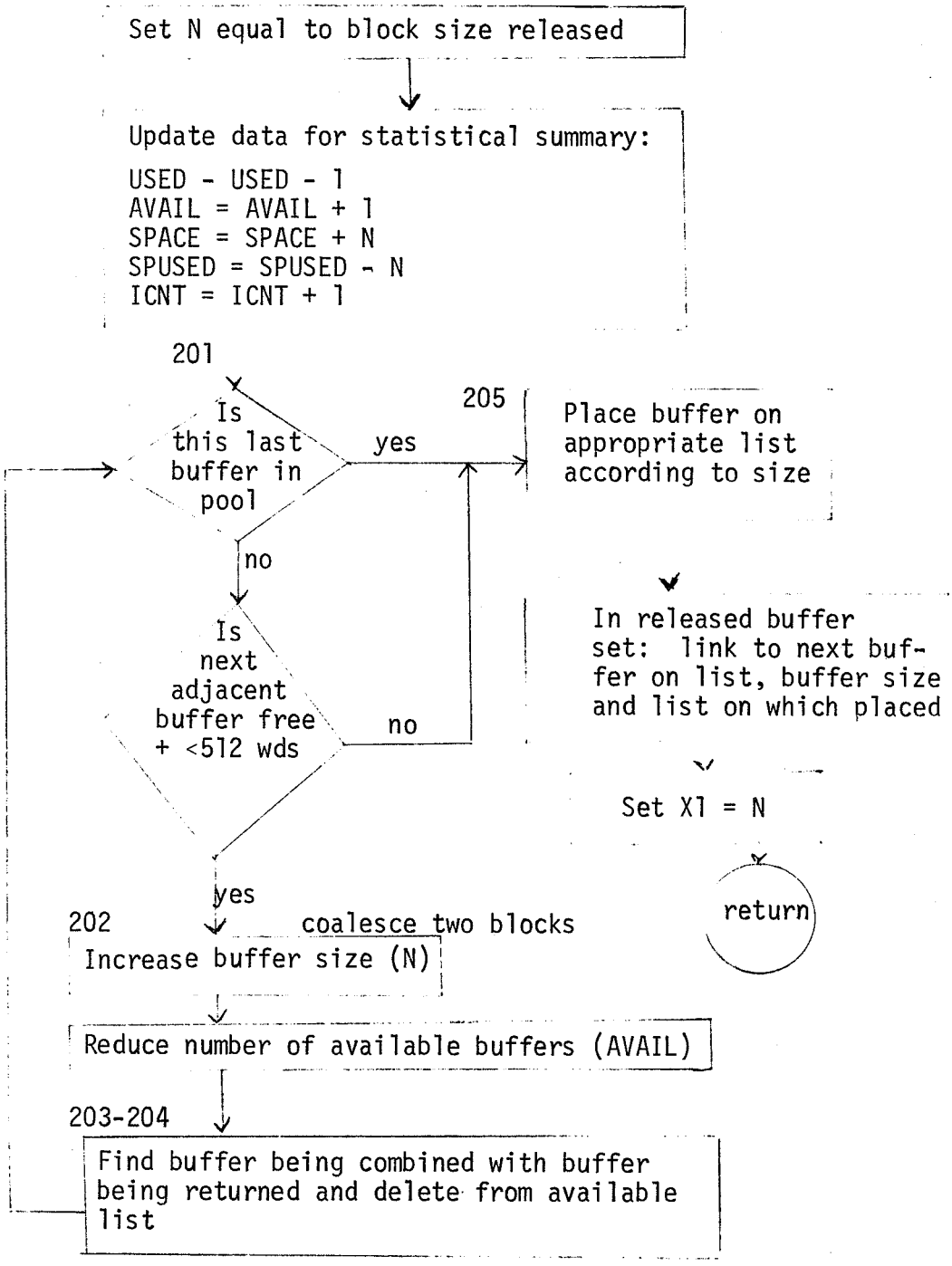


A-4. DETAILED OUTLINE OF HELP ROUTINE



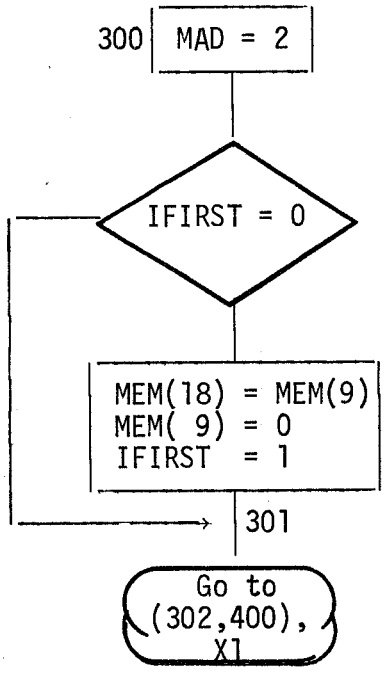
A-4 (CONTINUED)

First-Fit Release Routine



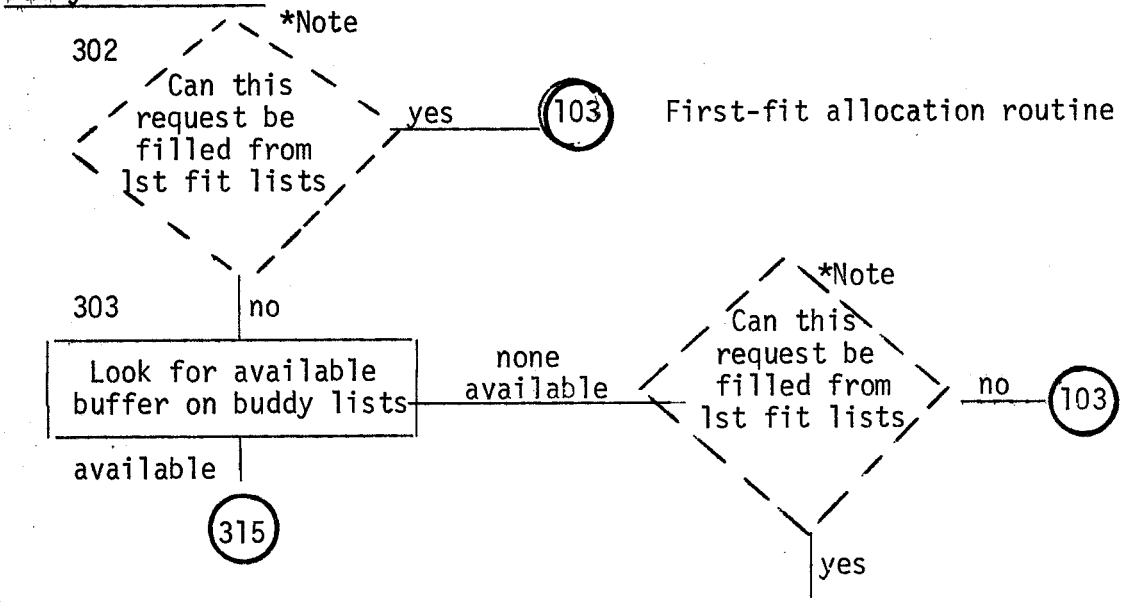
A-4 (CONTINUED)

Initialize Buddy Routines



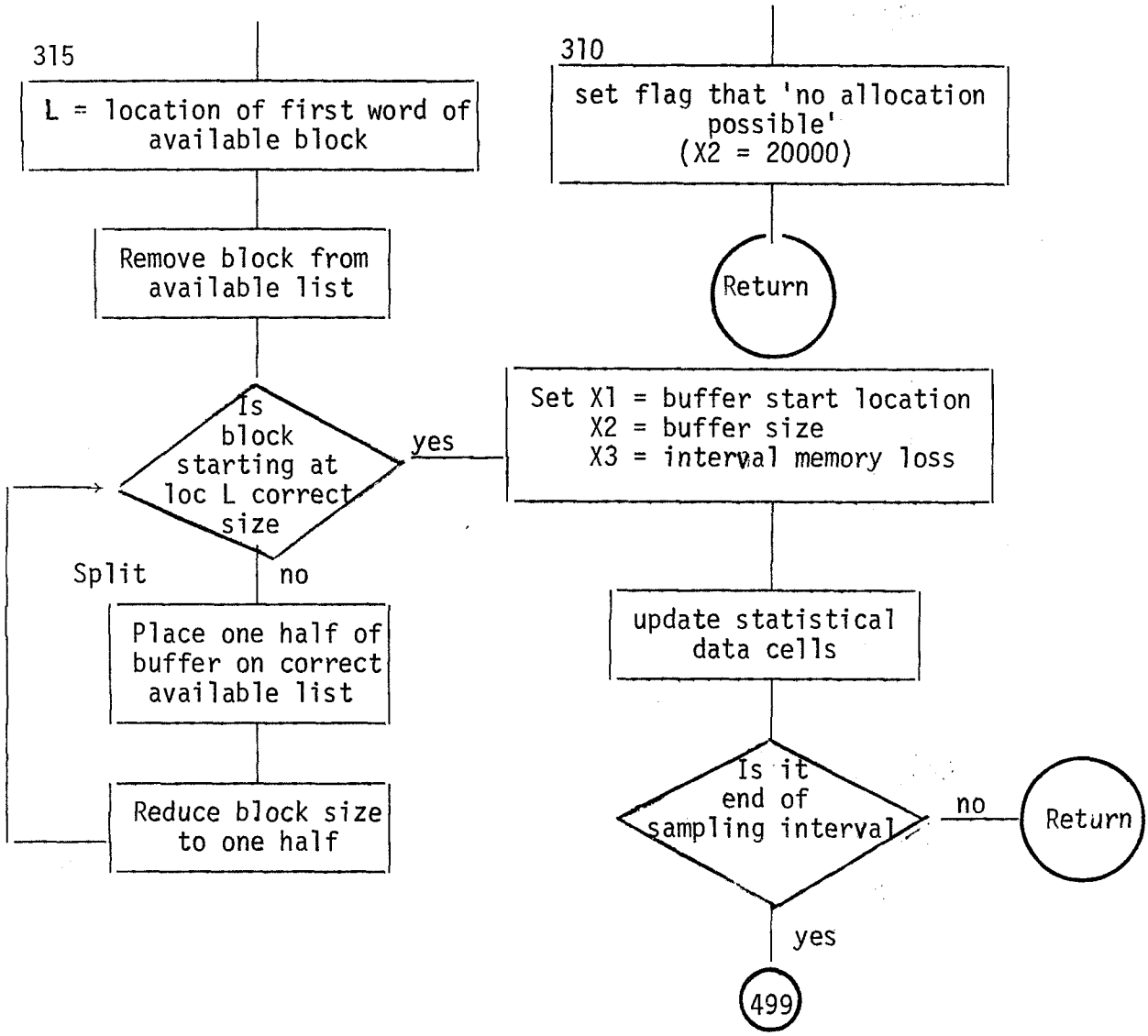
Move 2^9 blocks from first-fit list to buddy 2^9 list. Clear first-fit list, set flag to indicate initial entry to Buddy routine made.

Buddy Allocation



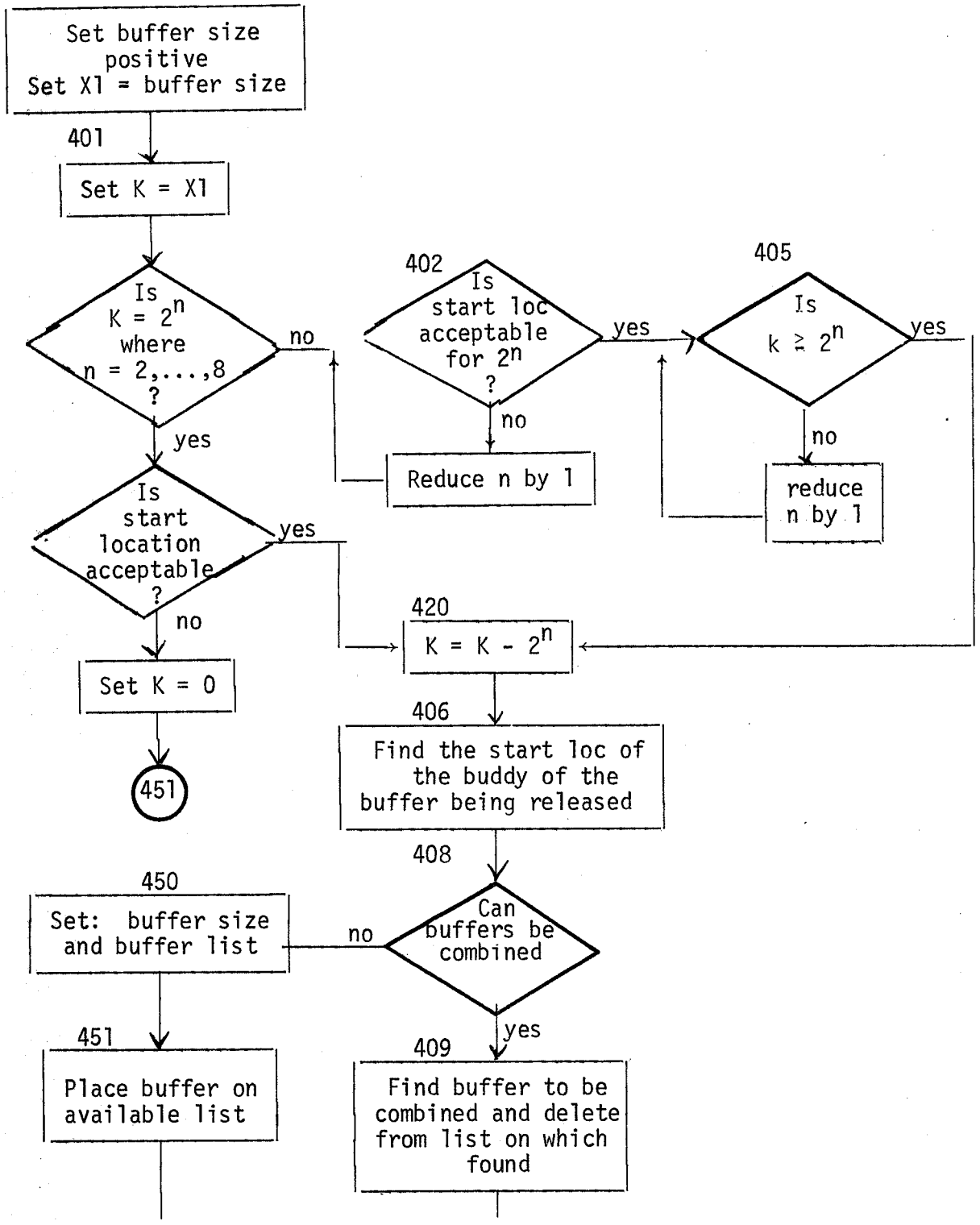
A-5. BUDDY ROUTINE

*Note: A search of the first-fit lists may be made at either of these points in the program. For any given run, only one is programmed.

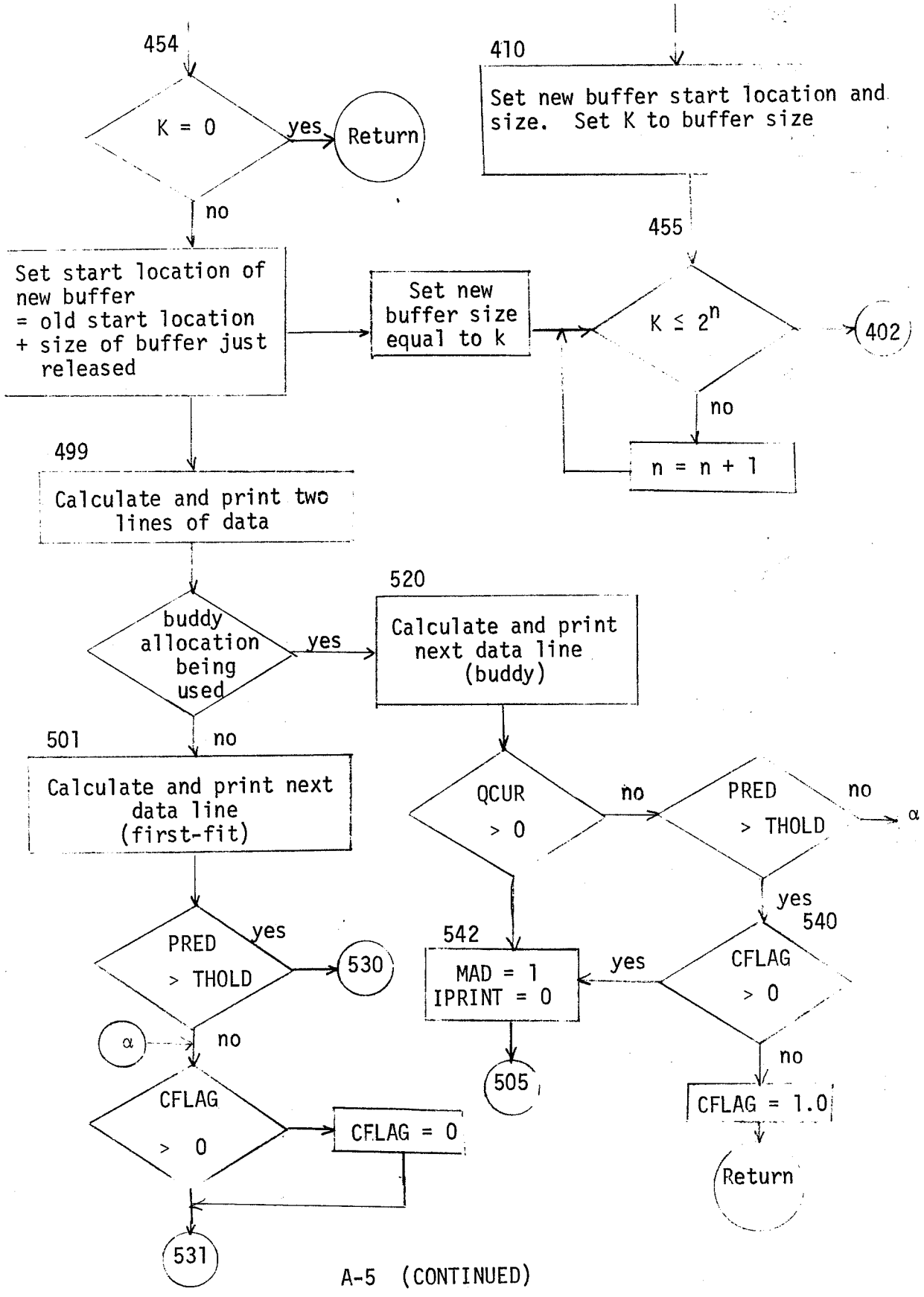


A-5 (CONTINUED)

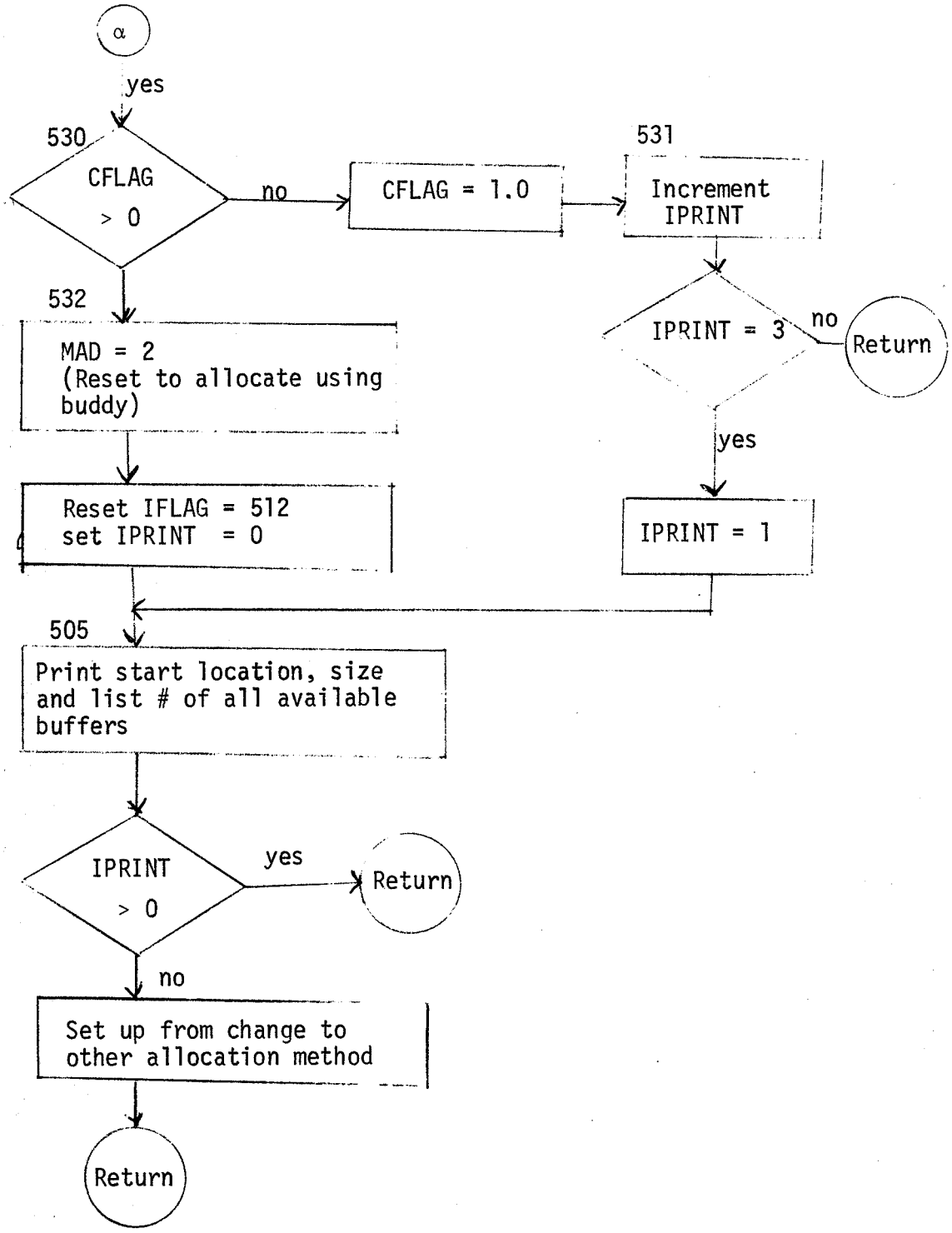
Buddy Release



A-5 (CONTINUED)



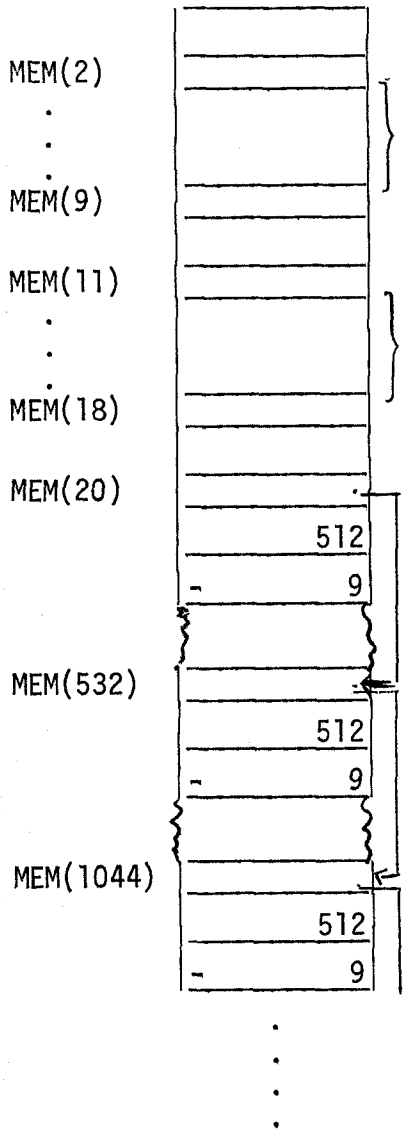
A-5 (CONTINUED)



A-5 (CONTINUED)

When the first-fit method of allocation is in use, all available buffers are accessed from MEM(2) to MEM(9). Lists headed by MEM(11) to MEM(18) are empty. Whenever the first-fit routines are entered after having allocated using the buddy method, the buffers from the buddy lists are placed at the end of any existing first-fit lists.

When the buddy method of allocation is in use the available buffers may be found on either the first-fit or buddy lists. In the allocation process, buffer requests are filled from the first-fit lists if possible. The buffers contained on the first-fit lists may be any multiple of four while the buffers contained on the buddy lists must be a power of two.

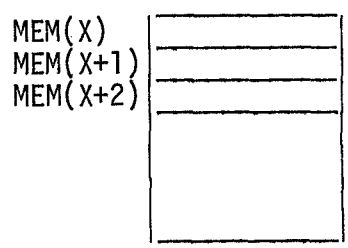


MEM(2) to MEM(9) contains pointers to lists of buffers of size 2^2 to 2^9 when first-fit method is in use

MEM(11) to MEM(18) contains pointers to lists of buffers of size 2^2 to 2^9 when buddy method is in use

Initially memory is broken into 512 word buffers and linked together. This list of buffers is accessed from either MEM(9) if first-fit is in use or from MEM(18) if buddy method is in use.

Each buffer has the following format.



MEM(X) is a pointer to the next available buffer on the list. If this is the last buffer on a list this location is set to zero.

MEM(X+1) Contains the buffer size. If the buffer is in use the sign is

negative. If the buffer is available, the sign is positive.

MEM(X+2) Contains the least power of two which is greater than the buffer size. If the buffer were placed on a list using the buddy method, the sign of this word is negative.

A-6 (CONTINUED)