B 1 8 0 0 / B 1 7 0 0

M C P M E M O R Y M A N A G E M E N T

CUBE XXXII -- Spring 1978

MCP MEMORY MANAGEMENT

INTRODUCTION

The B1800 and B1700 computer systems were designed to cover a rather broad range of the computer market. In order to cover this range with a single operating system, it was necessary to implement virtual storage capabilities and apply the same techniques used for normal-state programs to the operating system itself.

The main memory requirements for any computer system are highly dependent on an installation's applications and operating procedure. This fact is even more true of a variable-length segment, virtual storage system such as the B1800/B1700 which dynamically allocates memory to user programs as it is required. This type of system is able to keep many more programs in memory in order to provide higher processor utilization than are non-virtual systems or virtual machines with fixed page sizes ("partitions"). Since program segments are loaded only as needed, the memory requirements for programs on a machine such as the B1800/B1700 (and the B7800/B7700/B6800/B6700 systems as well) must be stated in terms of a "working set" rather than either total program size or minimum memory required to run.

The working set for a program is that amount of memory that it most often needs during its execution to operate efficiently. This working set must, of course, include the memory required for the functions requested of the operating system by the program, as well as certain operating system functions required for overall system control. The working set for the system as a whole is simply the sum of the working sets for all programs that are executed concurrently. If a program (or the system) is allocated less memory than its working set, it will demand non-present segments at a rate that will cause excessive segment overlaying and reduced efficiency. When the performance of the system degrades appreciably due to memory restrictions, the phenomenon is known as "thrashing".

CONCEPTS AND DEFINITIONS

MEMORY FRAGMENTATION

Fragmentation is the failure to allocate all of memory for useful purposes. Two varieties of fragmentation, internal and external, can occur. The type of fragmentation that will occur depends, respectively, on whether a system uses a paging or a segmentation mechanism for memory management.

In a paging system all of memory is divided into equal-sized pages or partitions. Therefore, 100% of memory is assigned to usable pages and external fragmentation does not occur. However, since memory requests typically are of varying sizes, the last page assigned to a memory request is usually not full. This is internal fragmentation.

In a system based on segmentation, segment sizes are variable so that only enough memory to satisfy a request is allocated to it. Therefore, no internal fragmentation exists in such a system. However, some memory is required as "overhead" for a memory link to describe each segment. A more serious problem is that an area of memory too small for use may become available between two segments of memory which are being used. This is external fragmentation.

Neither paging nor segmentation is clearly superior to the other. Each has its advantages and disadvantages. The primary advantage of paging is that it is straightforward from the point of view of the operating system. Segmentation, on the other hand, provides a much more reasonable structuring of memory since only the space logically required for a given function is allocated to that request. Therefore, programmers need not be concerned with trying to structure their memory requirements into requests that are exact multiples of the system's "page size". And the fewer unnecessary details that programmers have to think about, the more quickly and accurately they can complete their actual tasks. Segmentation does, however, cause "geography" problems for the operating system because external fragmentation "checkerboards" memory.

Burroughs has traditionally opted to use segmentation in its approach to memory management, and the B1800/B1700 systems are no exception to this rule. Therefore, memory management on B1800/B1700 systems is concerned with the algorithms and problems of segmentation.

WORKING SET

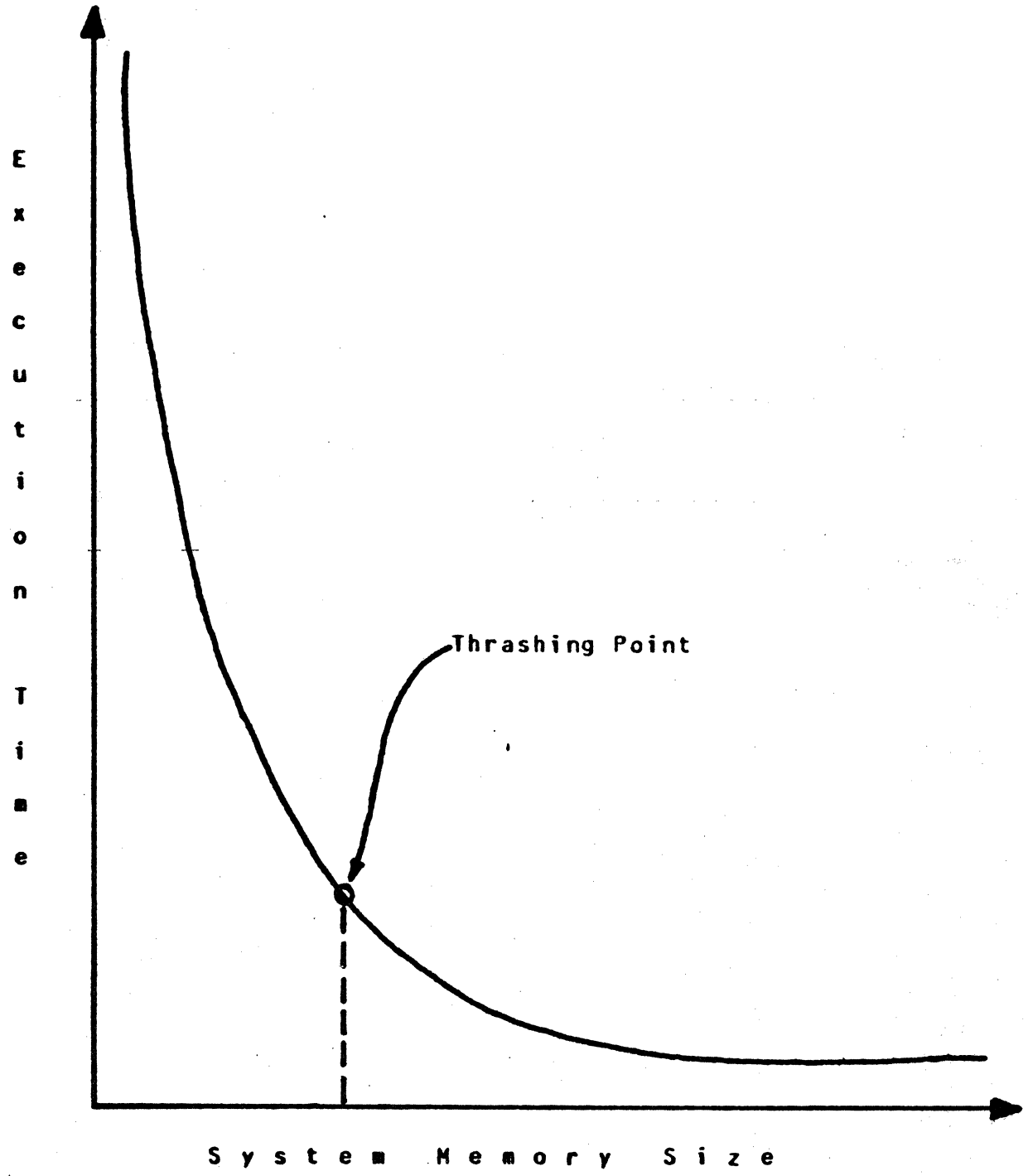
The term "working set" refers to the set of all program segments which are accessed during a specific time interval (of arbitrary length). The working set for a specific program is the set of data (the Run Structure), files, and code segments which it uses during such a time interval, plus the memory requirements of the operating system necessary to perform program-requested functions (READ, WRITE, OPEN, CLOSE, and so forth). The working set for the entire system is simply the sum of the working sets of all currently active programs. The working set for a program, and especially for the system as a whole, can and often does change drastically over successive time intervals as jobs go from one phase of execution to another.

THRASHING

"Thrashing" is the condition which exists when the working set for a program or set of programs does not fit in real memory. Specifically, in order to bring in the next code segment for a program, the operating system has to overlay a currently active code segment. Then that segment has to be brought back in, and another active segment must be overlayed, and so forth.

One of the most serious problems confronting virtual storage systems is thrashing. As the amount of memory available for a constant programming task is reduced, the amount of degredation due to thrashing normally appears very gradual at first. As the available memory is further limited (by introducing additional programs into the system, opening files, requesting additional or larger code segments, and so forth), a point will be reached where the degredation due to overlays increases rapidly. This is the point where the procedures in the working set no longer fit in memory and are competing for space. This point, referred to as the "Thrashing Point", is shown graphically in Figure 1.

System performance suffers drastically when thrashing occurs. Throughput degredation of 100% and over is not unusual in such instances. In fact, in the worst case absolutely nothing gets done except overlays.


Execution Time
Thrashing Point
System Memory Size

B1800/B1700 MEMORY MANAGEMENT ALGORITHMS

GENERAL

No single memory management system is ideal for all situations. Consequently, the B1800/B1700 MCP implements memory management on three separate levels of sophistication, using two different algorithms, in order to minimize the impact of more complex memory management schemes on those installations that do not need or want it. Installations that are satisfied with lower levels of the memory management system need not be concerned with the details of higher levels. This approach also allows users to ease into the more complex aspects of the memory management system smoothly, without being forced into an "all-or-nothing" decision.

LEVEL ONE (FIRST-IN, FIRST-OUT)

The algorithm of Level One is basically a "round robin" (or first-in, first-out) memory management scheme. When "available" memory space large enough to fulfill a request cannot be found by the MCP, one or more segments of in-use memory must be deallocated (overlayed). Overlayable memory is allocated starting from a "left-off pointer" which is then updated to point to the next lower segment in memory. Thus, the left-off pointer sweeps from high to low memory addresses until it reaches the first memory link, at which time it starts over again from the last memory link.

"Save" memory space, which cannot be reassigned until explicitly returned by the program to which it is assigned (for example, FIBS and File Buffers), is allocated toward the high end of memory so that it will tend to be pushed together, thereby reducing the external fragmentation that such "save space" inherently creates.

Advantages

1. External fragmentation of memory is minimized since small available chunks of memory tend to be swept up and used as the left-off pointer sweeps through memory.

2. Although a simplistic decision about what segment to deallocate is made, this decision can be made quickly. This is a very important feature, because if enough memory is available to contain the working sets of the currently active programs, then the first priority of the memory management system is to get that working set in as quickly as possible.

Disadvantages

1. The most serious flaw of this level is that there is no straightforward method by which a system user or operator can determine when memory has been overcommitted ("thrashing").

2. The relative activity of a segment and whether or not it is currently in use is not considered (or even known) when deciding to overlay that segment. Therefore, code segments which are no longer in use may be retained while "active" segments are overlayed.

3. The priority of a program using a segment is not considered when deciding to overlay that segment. Therefore, code segments of high-priority jobs are not protected from being overlayed by segments of lower-priority jobs. High-priority data comm jobs are a prime example of programs which often suffer because their segments are not protected from "background" jobs.

LEVEL TWO (FIRST-IN, FIRST-OUT WITH THRASHING DETECTION)

The second level of the memory management system implements detection for the "thrashing" condition. The same mechanism for determination of what segment to overlay (the "victim selector") is used for Level Two as for Level One. Thrashing detection is invoked following the next CLEAR/START operation by setting the "THR" MCP option, when SYSTEM/INIT incorporates the thrashing detection code into GISMO.

When GISMO, which is monitoring overlay activity, determines that thrashing is occurring and that it is not a temporary phenomenon, it notifies the MCP. The MCP then performs the following two functions:

1. Stops more programs from being automatically started. This can be overridden by the system operator by using the "PS" input message to "prod" the schedule. Otherwise the schedule will not be automatically restarted until some program goes to EOJ.

2. Sends the following message to the SPO:

   *** SYSTEM IS THRASHING, SCHEDULE STOPPED ***

   This message may be repeated either every time a program enters or leaves the MIX, or at every N.SECOND interval, as long as thrashing continues, depending upon the setting of the "THRASH" option of the "MM" input message.

When the system is shifting from one working set to another (as programs go to BOJ or EOJ, OPEN or CLOSE files, or move from one phase of execution to another), memory is often overcommitted for a short period of time. This condition is acceptable provided it does not persist for too long. One installation may, however, be willing to tolerate an overcommittment of memory for longer time intervals than another. For this reason, a means is provided (through the "THRASHING.SENSITIVITY" option of the "MM" input message) to adjust the sensitivity of the memory management system's thrashing detection mechanism.

In addition, the maximum overlay rate that can be tolerated is highly dependent upon the speed of the disk from which the overlays are being done, since more overlays can be performed efficiently during a fixed time interval from "fast" disk than from a "slow" disk. For this reason, a means is provided (through the "OVERLAY.RATE" option of the "MM" input message) to adjust this maximum allowable value.

The recommended value for the OVERLAY.RATE has been determined for the various disk types using their average access times (allowing for fixed MCP overhead required to obtain memory space and initiate the disk read), as shown in the following table:

Disk Type	Average Access	OVERLAY.RATE
B9480 cartridge	80 ms.	6
B9481 cartridge	100 ms.	5
B9482 cartridge	55 ms.	7
B9484 pack	33.5 ms.	10
B9499 pack	42.5 ms.	8
B9371 head-per-track	20 ms.	12
B9371 head-per-track	40 ms.	8
B9470 head-per-track	5 ms.	15

The default value for OVERLAY.RATE assigned by the MCP following a COLDSTART operation is ten (10).

Advantages

The advantage of this level is that system users and operators will know when their memory is overcommitted and will, therefore, be able to do a much better job of maintaining a mix of programs which utilizes most of memory but does not cause thrashing to occur.

Disadvantages

The only disadvantage of this level is that approximately 140 more bytes of non-overlayable memory are required beyond that of the Level One mechanism.

LEVEL THREE (MEMORY PRIORITY WITH THRASHING DETECTION)

This level of the memory management system includes the thrashing detection of Level Two, but a different "victim selector" based on job priority and segment usage. The Priority Memory Management algorithm is invoked following the next CLEAR/START operation by setting the "MPRI" MCP option, when SYSTEM/INIT incorporates the new "victim selector" into GISMO.

In this level, requests for segments of memory are assigned priorities which are separate and distinct from processor usage priorities (refer to the MEMORY.PRIORITY control instruction attribute and the MP input message). No request for memory may cause a segment having a higher memory priority to be overlayed.

In a mix with varying memory priorities, segments of high-priority jobs which are actively in use are protected from segments of lower-priority jobs. At fixed time intervals (known as the SAMPLING.INTERVAL), GISMO "sweeps" through all memory links on the system and examines a "usage" bit in each. This bit is set by a program's interpreter when the code segment is accessed (i.e., code in the segment is executed). If a segment has not been accessed since the previous "sweep" though memory, its priority is lowered by GISMO to the next lower memory priority active on the system. The segment is then protected at that priority for another SAMPLING.INTERVAL. If a segment is accessed at any time before being overlayed, it is restored to its original memory priority. In this way, segments of high-priority jobs are protected from those of low-priority jobs, and unused segments of any job tend to degrade to lower priorities and get overlayed.

In a "flat mix" (i.e., a mix with all memory priorities equal), those segments which are actively in use tend to stay in memory while those segments which are no longer being used tend to be overlayed. This cannot be made an absolute policy in a memory management scheme based on segmentation due to "geography" problems. For example, a very small inactive segment which has been allocated between two active segments may remain in memory longer than it otherwise would because of its location.

A "flat mix" has the additional advantage that it approaches the simplicity and efficiency of the Level Two algorithm as the system approaches thrashing.

The MCP sets the SAMPLING.INTERVAL value based upon the system memory size, as shown in the following table:

Memory Size	SAMPLING.INTERVAL
0-261 KB	8 (0.8 seconds)
262-523 KB	10 (1.0 seconds)
524 KB	12 (1.2 seconds)

A means is provided (through the "SAMPLING.INTERVAL" option of the "MM" input message) to change the rate at which the sweeper is executed, although changes from the default value should not be necessary and are not recommended.

Advantages

1. Varying memory priorities will protect active segments of higher-priority jobs from being overlayed by those of lower-priority jobs.

2. As in Level Two, the system operator will know when memory is overcommitted and will be able to do a much better job of maintaining a mix of programs which utilizes most of memory but does not cause thrashing to occur.

3. Running with equal memory priorities tends to make the system run in a manner approaching that of Level Two, with the added advantage that unused segments will degrade in priority and hence tend to be overlayed, while "active" segments will tend to stay in memory.

Disadvantages

1. Approximately 150 more bytes of non-overlayable memory are required beyond that of the Level Two mechanism.

2. If jobs are run at varying memory priorities, external fragmentation of memory can be increased.

EXTENDED SEGMENT DECAY

Level Three of the memory management system also allows protection of specified segments from overlay by lower-priority segments for an extended period of time (greater than the SAMPLING.INTERVAL) after they were last accessed. This capability is designed primarily to aid data comm installations which have no way of insuring that key segments of network controllers and other remote applications remain in memory. This problem can result in poor response time when low-priority "background" jobs cause data comm program segments to be overlayed. It is not advisable to permit such important segments to be marked as "save" (non-overlayable); however, Extended Segment Decay is only a little short of that capability.

There are two aspects to protecting key program segments:

1. Those segments which are to be protected for an extended period must be identified and marked. The means for accomplishing this is a utility program called "SYSTEM/MARK.SEGS".

2. Specification must be made of how long such segments are to be retained. This is done by setting the program attribute "SECONDS.BEFORE.DECAY" to some value between 0 and 600, inclusive (refer to the SECONDS.BEFORE.DECAY attribute).

The priority of segments which have been marked as important will not be degraded until and unless those segments are not accessed for the number of seconds specified by the SECONDS.BEFORE.DECAY attribute. It should be noted that if SECONDS.BEFORE.DECAY is set to zero for a particular program, then all of its code segments (both those marked as important and unimportant) will be treated as unimportant. Furthermore, SECONDS.BEFORE.DECAY is completely subserviant to memory priority. A segment with a higher memory priority can overlay a segment with a lower priority no matter what the value of SECONDS.BEFORE.DECAY for the lower-priority job. SECONDS.BEFORE.DECAY simply determines how long after a segment was last accessed it will be able to retain a given priority.

Note that specifying a SECONDS.BEFORE.DECAY value for a program that has no segments marked as "important" by SYSTEM/MARK.SEGS has no effect.

Advantages

Extended Segment Decay allows data comm users to guarantee that key segments of network controllers and other programs will not be overlayed by lower-priority jobs for any fixed period of time (between 0 and 600 seconds) after they are last accessed.

Disadvantages

Users of Extended Segment Decay can lock up more memory than they really need and thereby degrade the performance of background jobs more than necessary.

FUNCTIONAL CHARACTERISTICS

THRASHING DETECTION

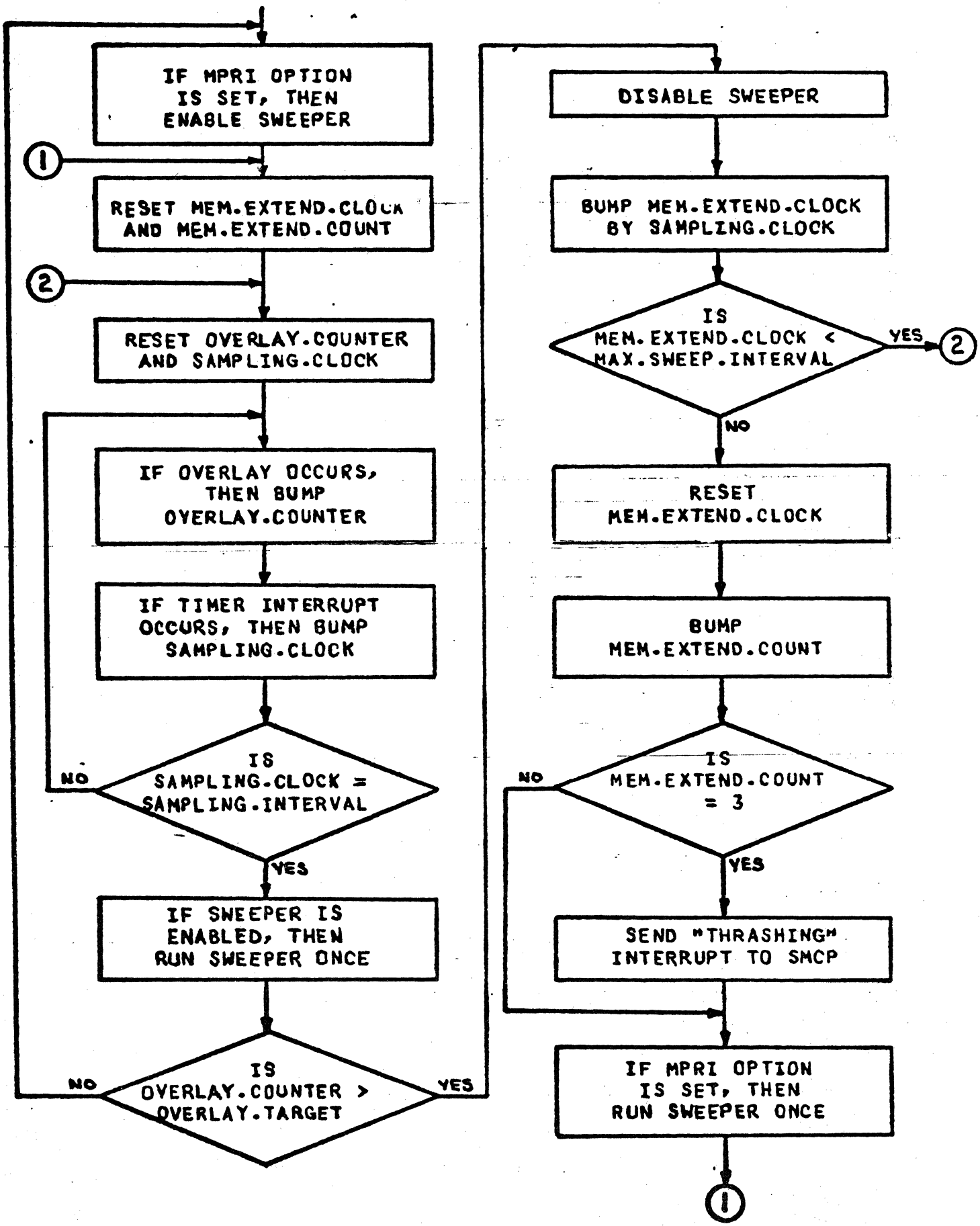
When thrashing detection has been requested (by setting either the "THR" or "MPRI" MCP options and performing a CLEAR/START), SYSTEM/INIT retains in GISMO the code necessary to monitor overlay activity.

The logic flowchart presented in Figure 2 depicts the general nature of the thrashing detection code. Certain data names have been used, some of which actually exist in the MCP and GISMO code. Other data names are fictitious, merely being used in the flowchart to represent a specific function. Their definitions are as follows:

SAMPLING.INTERVAL	A value (in tenths of seconds) computed by the MCP from the system memory size which specifies how often GISMO checks to determine whether thrashing is occurring. This value also specifies how often the "sweeper" is executed (if the MPRI option is set).
OVERLAY.TARGET	The value (in number of overlays per SAMPLING.INTERVAL) computed by the MCP from the specified OVERLAY.RATE and the SAMPLING.INTERVAL.
MAX.SWEEP.INTERVAL	A value (in tenths of seconds) computed by the MCP from the THRASHING.SENSITIVITY specified, equal to one-third of THRASHING.SENSITIVITY. This value also specifies how often the "sweeper" is executed once GISMO determines that the OVERLAY.RATE has been exceeded (if the MPRI option is set).
OVERLAY.COUNTER	A count of the number of overlays that have occurred. Reset to zero at the end of each SAMPLING.INTERVAL.
SAMPLING.CLOCK	A field that is incremented at each TIMER INTERRUPT until it reaches the value of the SAMPLING.INTERVAL.
MEM.EXTEND.CLOCK	A field that is incremented by the SAMPLING.CLOCK at the end of each SAMPLING.INTERVAL (if the OVERLAY.COUNTER exceeds the OVERLAY.TARGET) until it reaches the value of the MAX.SWEEP.INTERVAL.

MEM.EXTEND.COUNT	A counter that is bumped each time the MEM.EXTEND.CLOCK exceeds the value of MAX.SWEEP.INTERVAL. If this counter reaches a value of three (3), thrashing has continued for the length of time specified by the THRASHING.SENSITIVITY, and GISMO notifies the MCP of this condition.

References to the "sweeper" are applicable only if the MPRI option is set (refer to the following section on PRIORITY MEMORY MANAGEMENT).

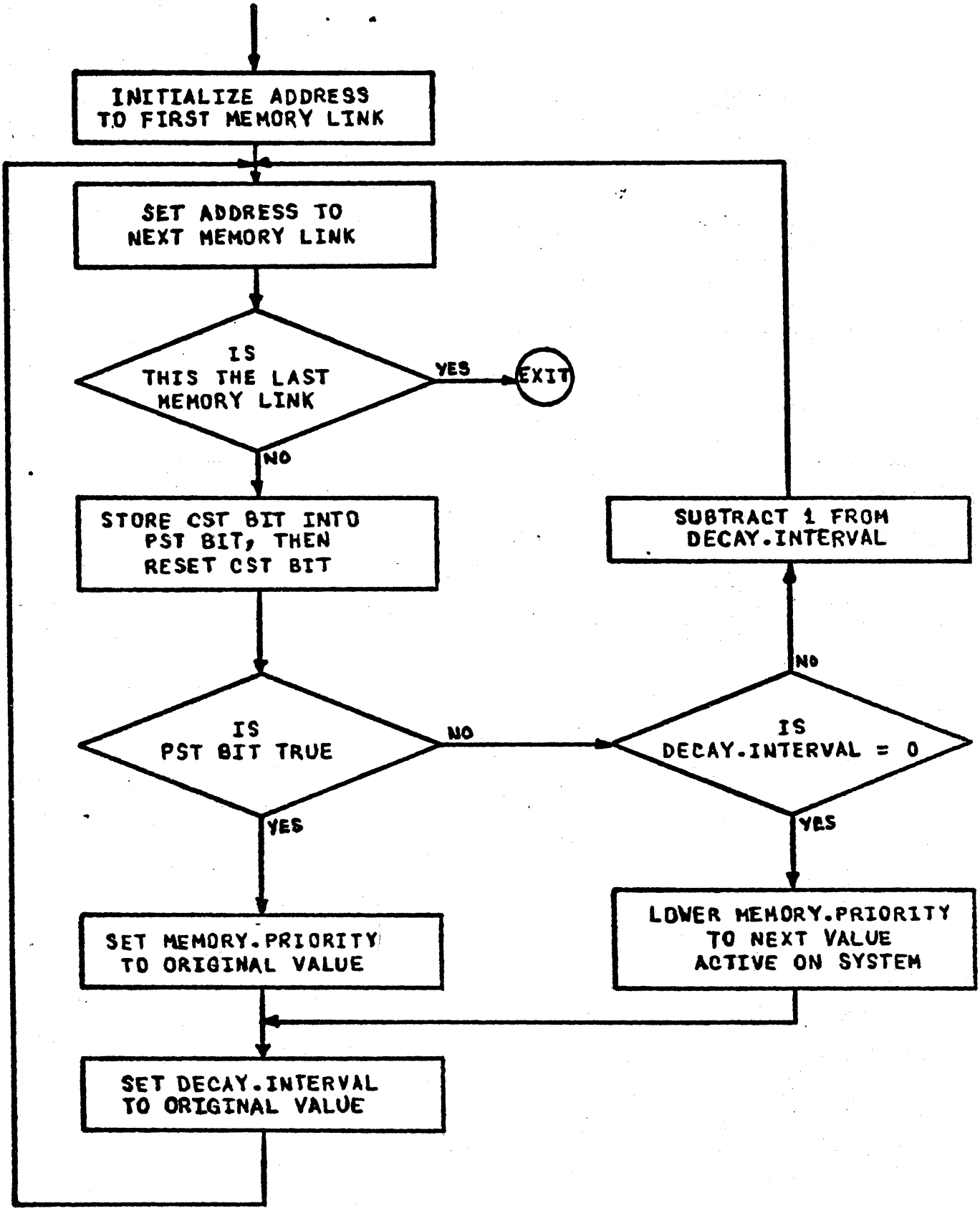

IF MPRI OPTION IS SET, THEN ENABLE SWEEPER
1
RESET MEM.EXTEND.CLOCK AND MEM.EXTEND.COUNT
2
RESET OVERLAY.COUNTER AND SAMPLING.CLOCK
IF OVERLAY OCCURS, THEN BUMP OVERLAY.COUNTER
IF TIMER INTERRUPT OCCURS, THEN BUMP SAMPLING.CLOCK
IS SAMPLING.CLOCK = SAMPLING.INTERVAL
NO
YES
IF SWEEPER IS ENABLED, THEN RUN SWEEPER ONCE
IS OVERLAY.COUNTER > OVERLAY.TARGET
NO
YES
DISABLE SWEEPER
BUMP MEM.EXTEND.CLOCK BY SAMPLING.CLOCK
IS MEM.EXTEND.CLOCK < MAX.SWEEP.INTERVAL
YES
2
NO
RESET MEM.EXTEND.CLOCK
BUMP MEM.EXTEND.COUNT
IS MEM.EXTEND.COUNT = 3
NO
YES
SEND "THRASHING" INTERRUPT TO SMCP
IF MPRI OPTION IS SET, THEN RUN SWEEPER ONCE
1

PRIORITY MEMORY MANAGEMENT

The Priority Memory Management mechanism, in addition to providing the thrashing detection capability described earlier, allows "active" code segments to be protected from overlay by lower-priority code. In order to prevent the total takeover of memory by high-priority code, however, GISMO periodically "sweeps" through memory and lowers the priority of those code segments which have not been accessed since the last time the sweep was performed. In this manner, segments which are not actively used by high-priority programs will be "decayed" until they reach a point where they can be overlayed by lower-priority segments.

The logic flowchart presented in Figure 3 graphically depicts the process by which the SWEEPER in GISMO examines each memory link and decays the priorities of unused segments. Certain data names have been used, some of which actually exist in the GISMO code. Other data names are fictitious, merely being used in the flowchart to represent a specific function. Their definitions are as follows:

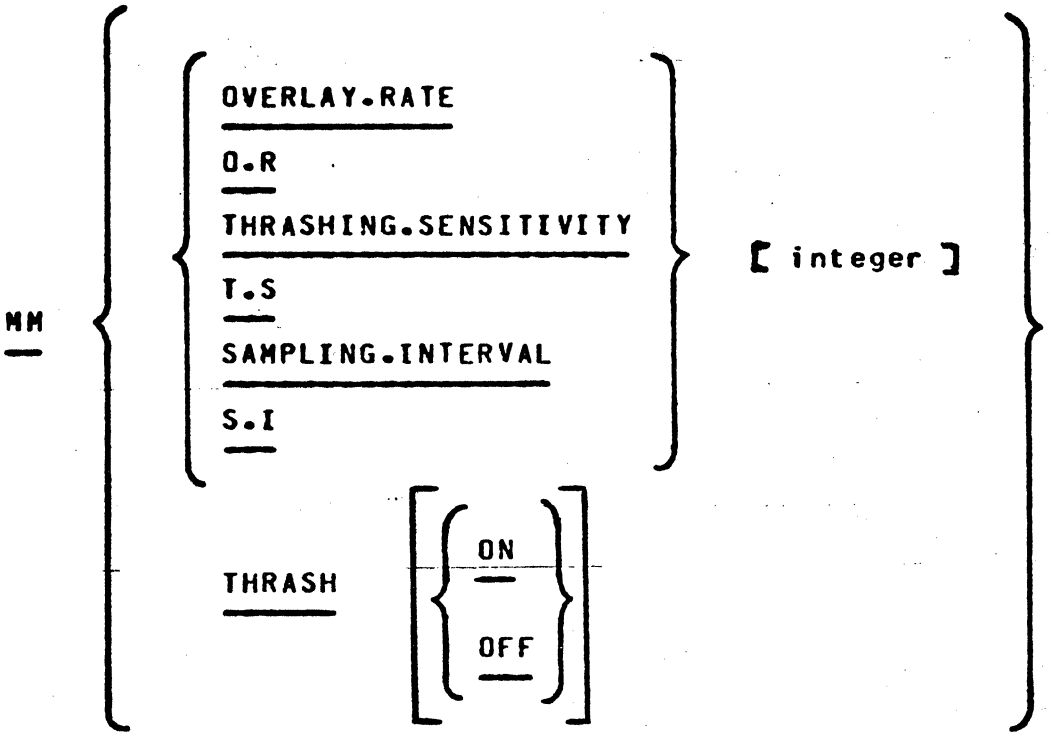
DECAY.INTERVAL	A value computed from the SAMPLING.INTERVAL and the SECONDS.BEFORE.DECAY specification which specifies the number of memory sweeps during which an unused segment will not be decayed in priority. For example, if the SAMPLING.INTERVAL is 8 (0.8 seconds) and the SECONDS.BEFORE.DECAY attribute for a program is set to 20, the DECAY.INTERVAL for all "important" code segments is set to 25. In other words, the code segment is protected from decay for 25 "sweeps" through memory (25 * .8 = 20). Code segments which have not been marked as "important" will always be marked with a DECAY.INTERVAL of zero (0).
MEMORY.PRIORITY	The value specified by the MEMORY.PRIORITY control instruction attribute or the MP input message.
CST and PST	Two bits in the memory link adjacent to a segment of memory that indicate its "in use" status. The CST (CURRENT.SCAN.TOUCH) bit is set by an interpreter whenever program control is passed to the adjacent code segment. The CST bit is reset only by the SWEEPER in GISMO. The PST (PREVIOUS.SCAN.TOUCH) bit contains the setting of the CST bit from the previous execution of the SWEEPER.


INITIALIZE ADDRESS TO FIRST MEMORY LINK
SET ADDRESS TO NEXT MEMORY LINK
IS THIS THE LAST MEMORY LINK
YES
EXIT
NO
STORE CST BIT INTO PST BIT, THEN RESET CST BIT
IS PST BIT TRUE
NO
YES
SET MEMORY.PRIORITY TO ORIGINAL VALUE
SET DECAY.INTERVAL TO ORIGINAL VALUE
IS DECAY.INTERVAL = 0
NO
YES
SUBTRACT 1 FROM DECAY.INTERVAL
LOWER MEMORY.PRIORITY TO NEXT VALUE ACTIVE ON SYSTEM

MM INPUT MESSAGE (Memory Management)

The MM input message allows the system operator to control certain attributes of the MCP Memory Management System.

The format of the MM message is:



The MM input message is not allowed if Level One (First-in, First-out) of the MCP Memory Management system is in use (i.e., neither the THR nor the MPRI option is set).

The OVERLAY.RATE (abbreviated O.R) option is specified in overlays per second, and may be set to any value between 1 and 20, inclusive. The default value following a COLDSTART operation is 10.

The THRASHING.SENSITIVITY (abbreviated T.S) option is specified in seconds, and may be set to any value between 10 and 60, inclusive. The default value following a COLDSTART operation is 20.

The SAMPLING.INTERVAL (abbreviated S.I) option is specified in tenths of seconds, and may be set to any value between 1 and 50, inclusive. The default value is set by the MCP during CLEAR/START, and is dependent upon the system memory size. This default value may not be changed by the MM input message unless the DBUG option is set (changes to the default value are not recommended).

The THRASH option specifies the frequency that the MCP will display the "SYSTEM IS THRASHING" message when the "thrashing" condition has been detected by GISMO. "ON" specifies that the message will be displayed at each N.SECOND interval (a variable period of time determined by the number of programs in the mix) as long as thrashing continues. "OFF" (the default setting) specifies that the message will be displayed by the MCP as long as thrashing continues, but only when a program enters or leaves the mix.

The values assigned to all options (except for the SAMPLING.INTERVAL) are retained by the MCP when a CLEAR/START is performed, and need not be specified again.

Omitting the value of any option (the integer or "ON"/"OFF") causes the current value of the option to be displayed by the MCP.

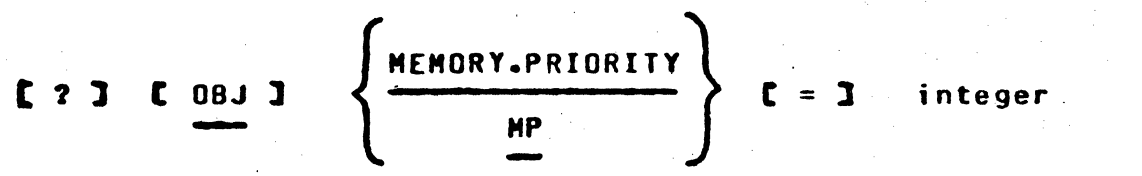
Examples:

```
MM O.R
 OVERLAY.RATE = 10

MM TRASH ON
 THRASHING.MESSAGE ON

MM T.S 15
 THRASHING.SENSITIVITY = 15
```

MEMORY.PRIORITY

The MEMORY.PRIORITY attribute allows specification of the priority to be assigned to segments of memory occupied by program code.

The format of the MEMORY.PRIORITY statement is:



The MEMORY.PRIORITY control word may be abbreviated as MP.

The MEMORY.PRIORITY attribute is only allowed when the "Priority Memory Management" algorithm is being used by the MCP (MPRI option is set).

The integer may be assigned a value from zero to fifteen (0-15), where zero is the lowest priority and fifteen is the highest.

When a program code segment is "made present" (read into memory) by the MCP, the memory space it occupies is given an initial priority equal to the MEMORY.PRIORITY of the program. Code segments of one program may not overlay those of another program which have a higher memory priority, thus allowing more important program code to be protected. However, code segments that are not referenced by a program for a period of time (equal to 1.5 times the SAMPLING.INTERVAL on the average, unless the SECONDS.BEFORE.DECAY attribute specifies a different interval for segments marked as "important") "decay" to lower memory priorities, thus eventually allowing them to be overlayed. If a segment is accessed by a program at any time before being overlayed, its priority is restored to the original value.

A MEMORY.PRIORITY value of nine (9) or greater is referred to as a "crashout" priority, and has a number of additional effects. If insufficient overlayable memory space for a request having a crashout priority is available, the MCP will attempt to deallocate "save" memory space having a lower memory priority. Such a deallocation is performed on the RUN STRUCTURE (BASE-LIMIT space) of a lower-priority program, and results in an abbreviated ROLLOUT of the program selected as the "victim". This action by the MCP, termed "crashout", suspends the "victim", and writes out to temporary disk storage only the program's BASE-LIMIT space (not any file or code space), then makes the space occupied by the RUN STRUCTURE available to satisfy the memory request. The MCP periodically (at each N.SECOND interval) attempts to reinstate any "victim" programs that were crashed out.

Note that entering a program having a crashout priority in the ACTIVE SCHEDULE will not cause any crashout actions to be taken on running programs in order to begin the high-priority job. Crashout can only be caused by an executing program having a memory priority of nine or greater, and whose memory priority is higher than that of the program which is to be crashed out (for example, a program with a memory priority of 12 cannot cause crashout on any other program with a memory priority of 12 or above, but can cause any program with a memory priority of less than 12 to be crashed out).

Example:

```
EXECUTE A/B MP=8
 A/B =3702 BOJ. PP=4, MP=8 TIME = 12:25:37.2
```

PROCESSOR.PRIORITY

The PROCESSOR.PRIORITY attribute allows the system operator to specify the priority to be assigned to processor usage by a program.

The format of the PROCESSOR.PRIORITY statement is:

[?] [OBJ] { PROCESSOR.PRIORITY / PP } [=] integer

The PROCESSOR.PRIORITY control word may be abbreviated as PP.

The PROCESSOR.PRIORITY attribute is only allowed when the "Priority Memory Management" algorithm is being used by the MCP (MPRI option is set).

The integer may be assigned a value from zero to fifteen (0-15), where zero is the lowest priority and fifteen is the highest.

If the PROCESSOR.PRIORITY of a program is set to nine (9) or greater, the following actions take place:

1. The SCHEDULE.PRIORITY is set to the same value as the PROCESSOR.PRIORIY (up to a maximum of 14), unless explicitly set to some other value in the COMPILE or EXECUTE control instruction (using the SCHEDULE.PRIORITY control instruction attribute).

2. The program is not considered by the MCP in determining whether or not the MIX LIMIT has been reached. The MIX LIMIT controls only those programs having a PROCESSOR.PRIORITY less than nine (9).

3. If the Priority Memory Management algorithm is not being used (i.e., the MPRI option is reset), the "crashout" capabilities provided by a memory priority of nine (9) or greater are associated instead with the processor priority (assigned using the PRIORITY control instruction attribute or the PR input message).

Example:

```
EXECUTE A/B PP=8
 A/B =3702 BOJ. PP=8, MP=4 TIME = 12:25:37.2
```

SECONDS.BEFORE.DECAY

The SECONDS.BEFORE.DECAY attribute allows specification of the length of time to protect unreferenced code segments marked as "important" from being degraded in priority by the MCP.

The format of the SECONDS.BEFORE.DECAY attribute is:

[?] [OBJ] { SECONDS.BEFORE.DECAY / SB } [=] integer

The SECONDS.BEFORE.DECAY control word may be abbreviated as SB.

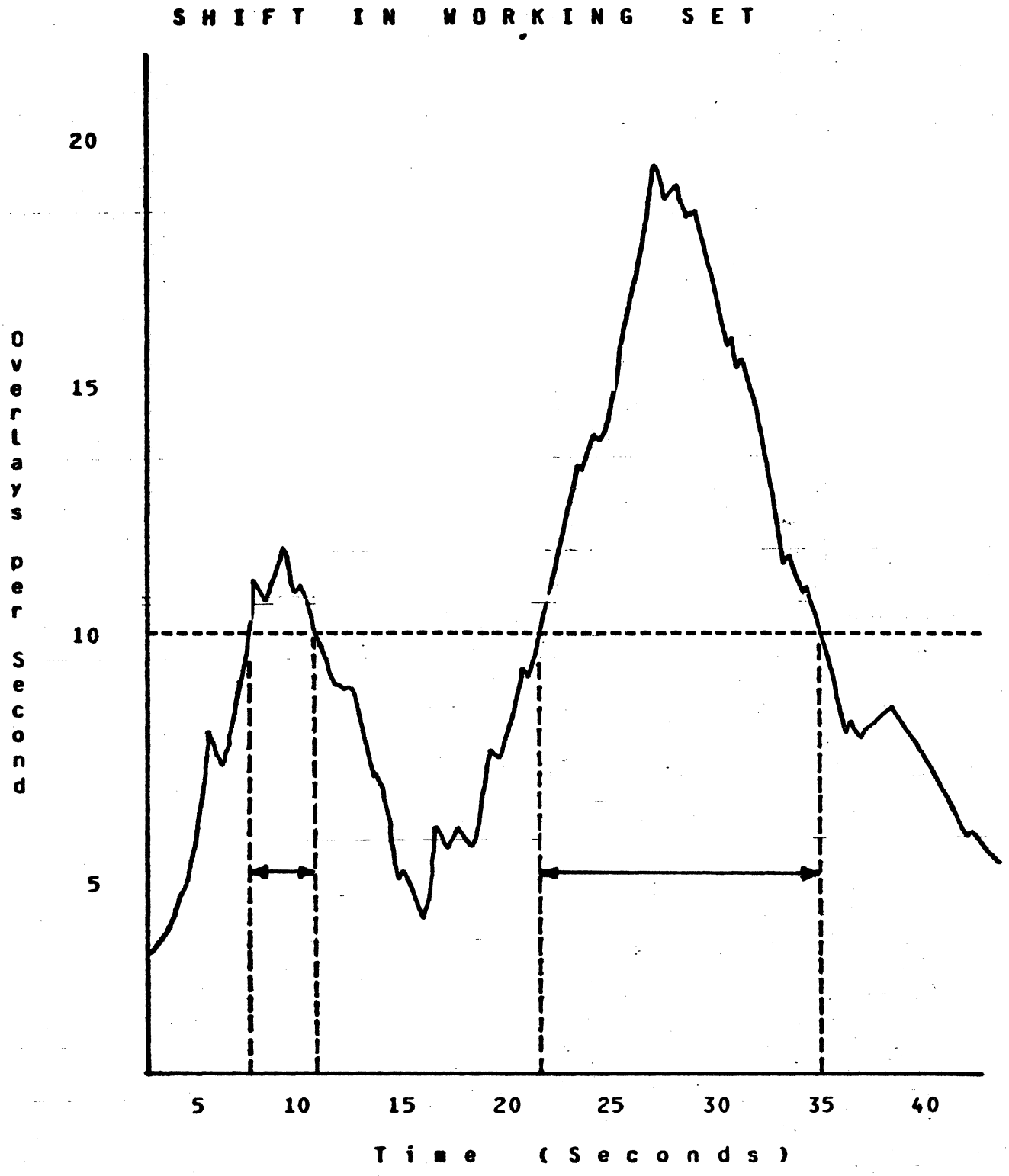
The SECONDS.BEFORE.DECAY attribute is only allowed when the "Priority Memory Management" algorithm is being used by the MCP (MPRI option is set).

The integer may specify any value between 0 and 600, inclusive, and designates the length of time (in seconds) that an unreferenced code segment which has been marked as "important" is to be retained at its current memory priority before being degraded ("decayed") to a lower priority. If the value of SECONDS.BEFORE.DECAY is zero, all code segments of a program, whether marked as "important" or not, are treated as "unimportant" (that is, Extended Segment Decay is not applied).

A special system utility program, SYSTEM/MARK.SEGS, must be used to mark specific code segments as "important" for use with the SECONDS.BEFORE.DECAY attribute. Specifying a non-zero SECONDS.BEFORE.DECAY on a program which has no code segments marked as "important" by SYSTEM/MARK.SEGS has no effect.

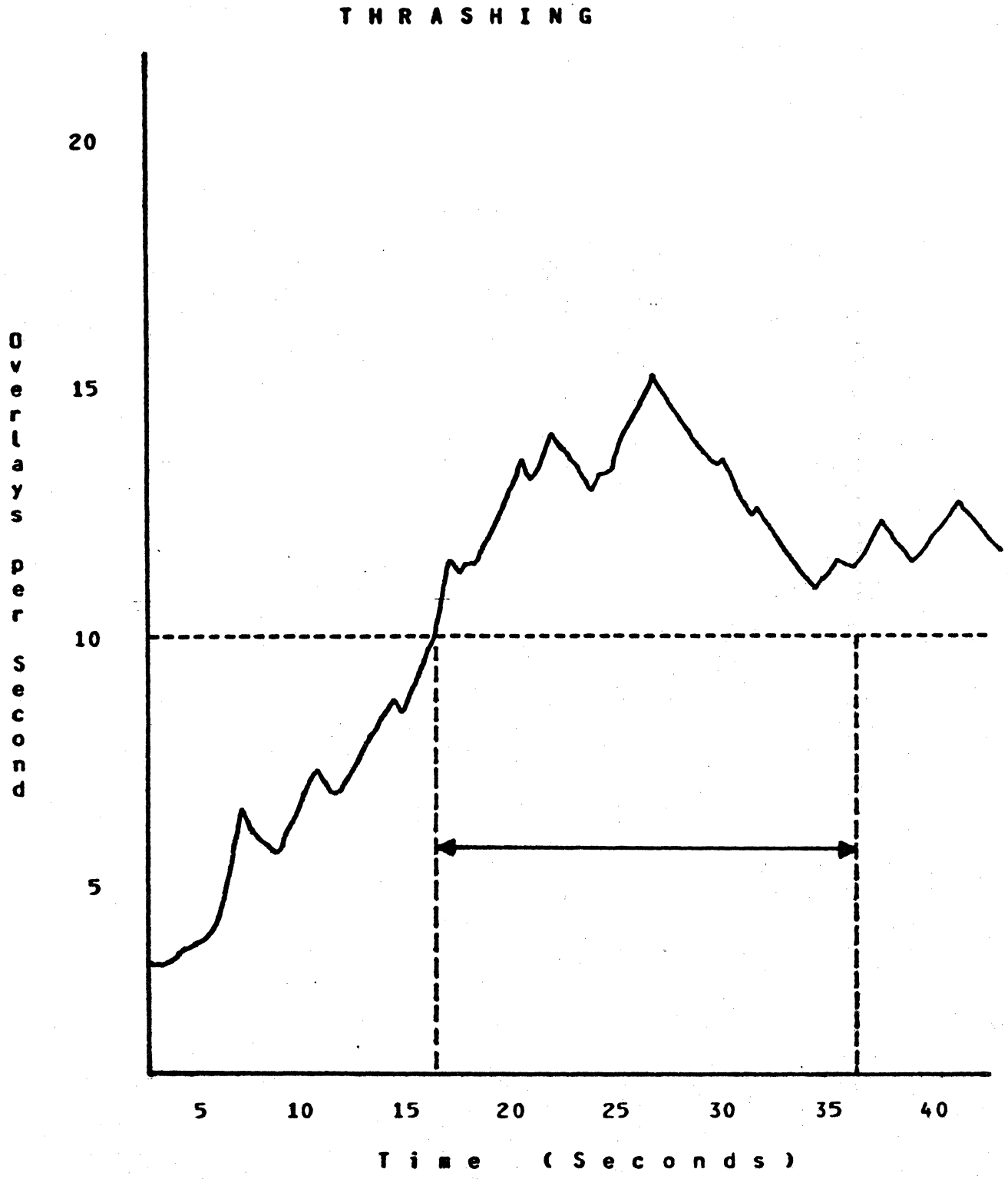
Example:

```
EXECUTE A/B MP=15 SB=250
```



OVERLAY.RATE = 10

THRASHING.SENSITIVITY = 20



OVERLAY.RATE = 10

THRASHING.SENSITIVITY = 20