ISSN 0381-9019

A PARTITION MONITOR FOR FAST-BATCH-PROCESSING WITH LIMITED EXECUTION (FABLE)

BY

PATRICK P. EMIN

TR76-012, MARCH 1976

A PARTITION MONITOR FOR FAST-BATCH-PROCESSING WITH LIMITED EXECUTION

(FABLE)

BY

Patrick P. Emin

Computing Centre
and
School of Computer Science
University of New Brunswick
Fredericton, N.B.

TR76-012, March 1976

ABSTRACT

FABLE is a sub-tasking monitor which was designed originally to execute as a problem program under IBM's OS/MFT-II (with sub-tasking) operating system. It provides a facility for programmers at any HASP workstation, reader or RJE terminal, to submit programs or problems to a variety of compilers and processors, such as: WATFIV, PL/C, ALGOLW, WATBOL, and ASSIST, without the usual overhead required for job scheduling and resource allocation. Output is returned to the same terminal with very fast turnaround. Programs are submitted in batches of one or more problems, for one or more processors, in any order; the only job control language (JCL) required is one job card per batch.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

Many thanks are extended to Dr. D.M. Fellows and Professor B.J. Claus for their valuable assistance in the initial stages of problem analysis. Also to Messrs. Brian Cassidy and Brian Kaye of the Computing Centre, whose unfortunate task it became to program and implement FABLE - with admirable results.

PREFACE

The FABLE system design as presented in this paper, was adopted for use at the University of New Brunswick Computing Centre in the summer of 1972, upon agreement among the member installations of the New Brunswick Educational Computing Network. The only feature not required at that time was the ability to account for and restrict resource utilization (c.f.SALT).

Since its implementation during the fall of the same year, it has been adapted to a number of different IBM operating systems, and has undergone a number of minor changes in addition to a design modification dealing with the problem of core fragmentation.

FABLE was originally interfaced with OS/MFT-II (with sub-tasking), release 20. As its use increased, so did the incidence of core fragmentation which could seriously reduce the amount of usable free memory available (and guaranteed) to the user. This was due mainly to the fact that some processors fail to close their active files at termination, in addition to various other control blocks and tables not being freed. As a result, the availability of core memory was monitored via a GETMAIN SVC under the MFT system, and in 1973 under OS/MVT release 21, and steps taken to recover unusable fragments. However, with the advent of IBM's virtual storage system, OS/VS2 release 1, a more efficient operating system, it was deemed advisable to alter the FABLE design as follows: the method of attaching one sub-task (c.f.MODLIN) only, which in turn loads and deletes each processor as required, was abandoned in favour of directly attaching and detaching each processor as a sub-task. This latter method has the advantage, at detach time, of closing all data sets, concelling various data elements and freeing a main memory storage pool. However, measurements indicated that system processing time overhead increased by a factor of six for each ATTACH/DETACH of a processor.

Other enhancements to FABLE included an interface for high-level language processors (usually user-written) such as PL/1, in the form of user-callable subroutines, another being the ability to attach virtually any program module (however, to date, this has not been implemented as a FABLE service).

What lies in the future for FABLE will probably be determined primarily by sheer necessity. However, one feature which the author proposes, is taking more advantage of the concepts inherent in the virtual storage system, such as: having more than one processor active at a time, (i.e., increased multi-tasking) with their address spaces paged between main and auxiliary storage as required. This would take advantage of the more efficient hardware assist used for transferring data through I/O channels, in lieu of the costly overhead now associated with the frequent program fetch. In addition, the ATTACH/DETACH degradation is substantially reduced, being necessary only at initialization and on the rare occurrence of sub-task abnormal termination.

Although FABLE was designed to accommodate additional multi-tasking through its scheduling information tables, interprocess communication primitive and elements (c.f. SIT, SHED and PRICE), a more direct, simple and efficient method might be the use of P and V operators with counting semaphores (see for example: "Operating Systems - Madnick and Donovan, 1974")

The race condition which would result from the shared SPOOL output, is resolved by the simple expedient of defining separate output files for each sub-task and modifying the processor file names accordingly.

Patrick P. Emin
March 15, 1976

A PARTITION MONITOR FOR FAST BATCH WITH LIMITED EXECUTION (FABLE)

DESIGN PROPOSAL

June 19, 1972

1. INTRODUCTION

Many university computing centres have in operation at their installation, one form or another of a multi-batch switching monitor, which executes in a partition (or region) of the Operating System, and depending on requests received through the input job stream, will load, link to or attach one of several different batch compilers/processors to service the users' requests. [1]

Such monitors are custom-designed for a particular installation to meet individual requirements of both the installation administration and the users. Some of the distinguishing features found in various monitors are:

- Ability to switch execution from one type of batch processor to another upon recognition of a header record peculiar to each processor.
- Very little control program or other overhead associated with the switching process.
- Ability to intercept abnormal termination of a processor from within the monitor, with a minimal loss of input job stream.
- Ability to provide accounting statistics in relation to users of the facility.
- Ability to impose processing limits and restrictions upon users, to ensure fast turnaround.
- Ability to introduce jobs into the monitor's input stream from various sources, i.e., terminals.

Not all of these characteristics may be found in any one monitor; however, most of them require extensive changes to the individual processors which makes it difficult to add new ones, and some require installation dependencies which place too many constraints upon other prospective users.

It is with these latter observations in mind that the design outlined herein is based.

2. PROBLEM DEFINITION

At the University of New Brunswick Computing Centre, fast turnaround batch has, until now, consisted of the use of the WATFIV compiler. The requirement there, is for a method of supplying a similar service for PL/C, ALGOLW, SPASM in the immediate future, and also to provide for accounting of users, impose processing restrictions on individual jobs, and accept input from various sources. Also, it would be desirable to require as few changes to the individual processors as possible, and allow for the addition of new processors as required. In addition, a built-in facility for listing decks of cards on a printer is desirable.

3. SYSTEM DEFINITION AND ANALYSIS

The following discussion predisposes the recognition of five distinct but inter-related processes, each comprising a program module and a method of inter-module communication and process scheduling. The five processes are:

- A partition monitor and switching task (main task)
- A processor linkage task (sub-task)
- A processor module (one of several which may be loaded into main storage at any one time).
- An accounting module which will initially consist of a dummy routine for testing linkages and return codes only.
- A job stream flush routine. This will be used for two purposes:
  - To search the input stream for '??EXEC' records while listing unprocessed records on the SYSPRINT device.
  - To provide a card deck listing facility on reader-printer terminals.

Each of these will be treated separately as regards design and implementation. By the same token, each related program module may be coded and tested by one or more programmers concurrently, or at different times.

The expected job stream for the monitor as received from HASP consists of a series of sub-batches, each prefixed with a form of 'EXEC' card bearing a processor name and user accounting information, and terminated by an end-of-file condition (Refer to Figure 1). The sub-batches are each input for different processors, but consecutive sub-batches bearing the same accounting information belong to the same physical batch entered at a terminal by one user-group.

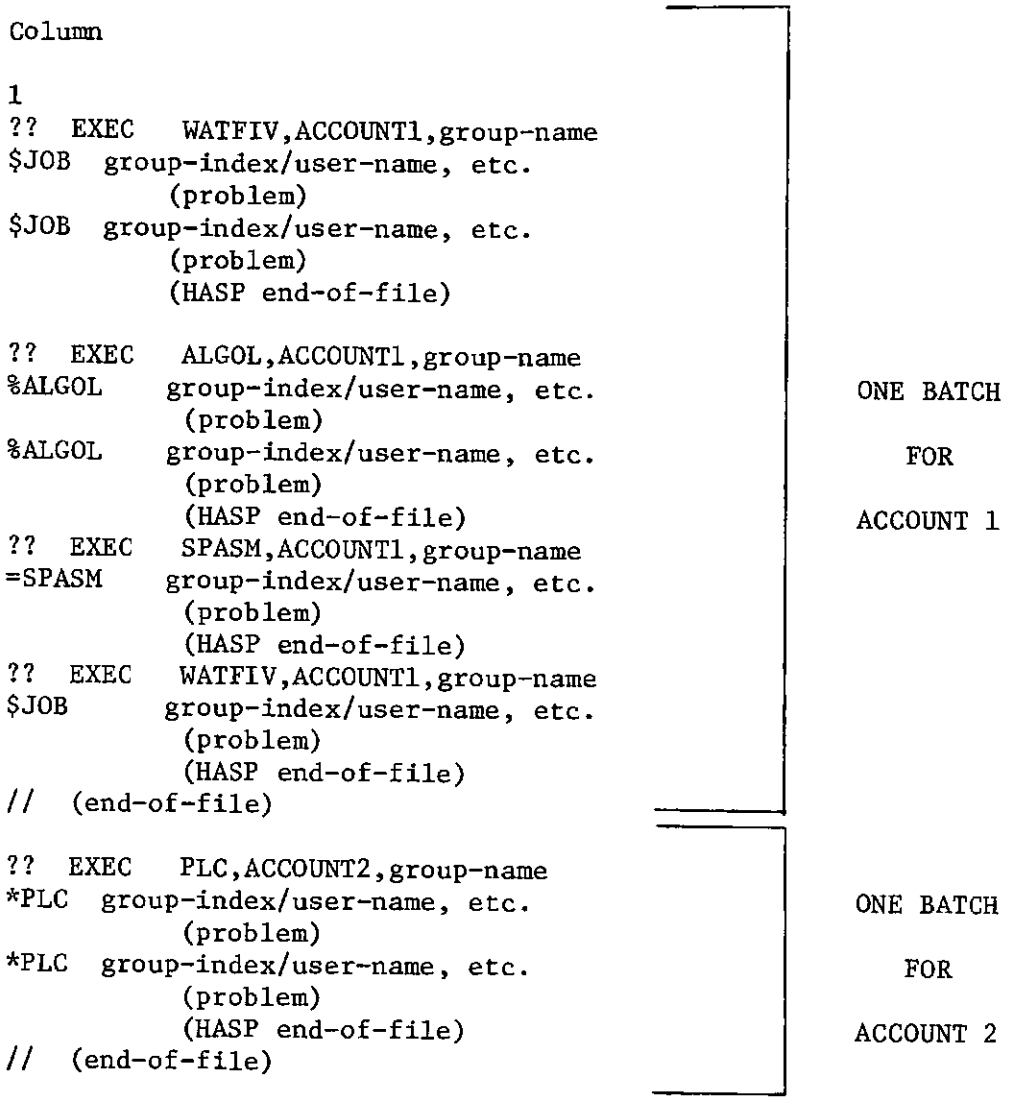


FIGURE 1 - FABLE JOB STREAM (from HASP)

Upon recognition of an '??EXEC' record, the monitor must determine if accounting is the same as previous, and if not, initiate a new accounting record, see if a new processor is required and LOAD it to main storage if required, "ATTACH" the required processor task and relinquish control to it.

The processor will terminate upon detecting an end-of-file from HASP, and control will return to an end-of-task exit routine (ETXR) in the monitor. While in control, a processor will input and process problem program jobs in the same manner as it would when executing normally under the OS control program.

4. PRIMARY DESIGN CRITERIA

The Monitor - following are the salient features of the required monitor (refer to Figure 2).

- Processes original parm list passed by control program.
- Constructs Processor Operating Table (POT) for each processor attached, using original parm list.
- LOAD and DELETE each processor as required, and execute under an attached TCB belonging to the module linkage routine (MODLIN). This should reduce some overhead imposed by control program when attaching. (It may even be possible to bypass DETACH after normal completion of processors, and simply ensure that processor required is in core, and re-enter MODLIN which is re-entrant - to be investigated).
- Provide various service routines within the monitor, i.e., accounting, flushing, etc., and passing their addresses to the processor via the Monitor Entry Address Table (MEAT).
- Provide an ETXR for the sub-task. This should obtain task info. via EXTRACT, check for normal/abnormal completion, etc., decide upon next course of action, i.e., close off accounting, flush SYSIN, re-attach old processor, etc., DETACH old task if necessary.
- Establish an inter-module/inter-process communication convention, with main task/sub-task event synchronization obtained by the use of macro-instructions which employ WAIT and POST. Scheduling of essential events within both monitor and processor will depend upon the alternation of dispatching between the monitor task and the processor task by the use of WAIT and POST. (See Appendix VI).

- Accounting must be done by a module to be prepared later, which will supply the following:

  - Provide OS Batch accounting records (BAR) by batches, using the accounting info. passed to it by HASP on the '??EXEC' record, and the cards, lines, etc. also obtained from HASP, together with I/O counts, times, etc., obtained by the easiest means. (e.g. by searching SMF tables)

  - Provide group accounting statistics (GAS) in the way of cumulative CPU time, problem count, normal ends, cards read, lines printed, etc. These values will be charged to a group of users, pointed to by a number found on the processor header record, which will be used to index into a table of accumulators, which must be updated at the end of each processor problem. At the end of batch, the cumulative values will be recorded on the accounting data set, in addition to the OS accounting record. Space for 30 groups is initially required, with each group requiring a maximum of 8 bytes for accumulators, i.e., a record 240 bytes long must be maintained, in addition to any other data which must be prefixed and/or appended to it. e.g., OS account number, dates, times, etc.

- Establish communication with HASP in order to obtain job information , i.e., cards read, lines printed, etc.

The Module Linkage Routine (MODLIN). This is a simple reentrant piece of coding which does nothing more than pass a parameter list to a processor, as it passes control to its entry point. Whenever it is ATTACHED, after the first time, it need not be brought into main storage, since it will already exist in the Job Pack Area of the partition. In addition, it will pass control to an existing copy of the required processor (refer to Figure 2).

The Processor - following are the required changes to individual processors:

- Process the parameter list provided by the monitor via the module linkage routine (MODLIN), or the control program, and making the necessary changes or additions to processor operating characteristics by using values passed via the POT.

- Isolate the following functions within each processor, and insert the necessary coding and/or macro calls to establish proper linkage with the monitor via the MEAT;

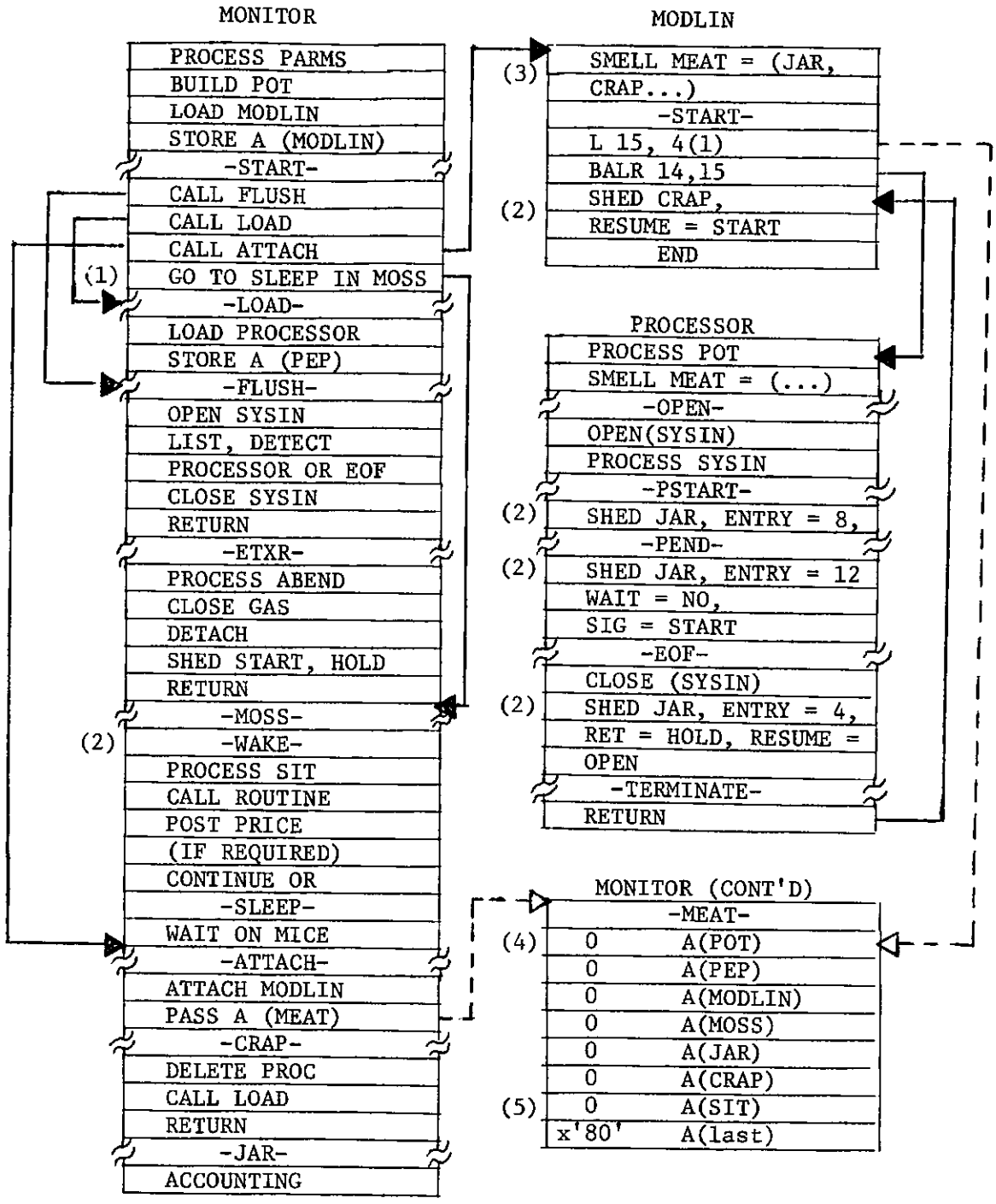


FIGURE 2 - Inter-process Parameter List Relationship and Control Transfer

Figure 2 (continued)

NOTES:

(1) After ATTACHing MODLIN, and after servicing processor requests, MONITOR will WAIT on MICE and remain dormant in MOSS(Multi-tasking Operations Synchronization and Switching) routine.

(2) MONITOR is activated at WAKE in MOSS by action of SHED macro. MOSS performs requested services through sub-routine linkages.

(3) Specify Multi-tasking Entry List Locations (SMELL) macro establishes user-specified symbolic entry points in MEAT.

(4) Processor Operating characteristics Table (POT) expanded in Appendix II.

(5) Scheduling Information Table (SIT) expanded in Appendix VI.

(6) LEGEND

——————→ control transfer

- - - - - -▷ pointer

- Problem program initiation:

  Pass accounting from processor header card for verification and group accounting (GAS), and increment processor job count accumulator.

- Problem Program Termination:

  Pass completion code to GAS routine and request updating of GAS accumulators. Provide GAS routine with problem statistics, i.e., CPU time, cards read, lines printed.

The Accounting Module - the desired functions and features are as follows:

- must have at least 4 entry points, or provide 4 distinct functions:

  - Entry code=0 - Delimits any previous accounting, initiates accounting for a new user. Initializes timers, counters, GAS accumulators, etc. for new account number. HASP has already verified the accounting field on the '//JOB' record, and transferred this info. to each '??EXEC' record in the batch. This routine entered from monitor only when new '??EXEC' record encountered. If previous accounting not terminated, after delimiting, its record must be written out to the accounting data set. Exits back to caller.

  - Entry code=4 - terminates current accounting, writes accounting record to external medium, returns to caller. Entered from ETXR indirectly after monitor posted for execution.

  - Entry code=8 - Entered from processor problem initiation routine. Verifies group account number. Accepts or rejects job. (In future, checks remaining time, cards, pages allocation also.)

  - Entry code=12 - Entered from processor problem termination. Accepts parameter list of CPU time, cards read, lines/pages printed, completion code, etc. Updates GAS accumulators.

The FLUSH - (Force Lines Until Search Honoured)

Reads records from SYSIN and lists on SYSPRINT until end-of-file or new '??EXEC' record encountered. Entered from various points in monitor and processor, and also as the result of processing a '??EXEC LISTER' record, which is constructed by HASP upon recognition of a '?LIST' job card followed by cards to be listed on the printer.

5. SYSTEM OPERATING CHARACTERISTICS

The Monitor. A preliminary concept of the various stages of operation, linkages, inter-communication, task-switching, etc. is described hereunder (refer to Appendix I for a system flowchart). Flow of control will be traced beginning with the initiation of the FABLE monitor as a job step under the control program.

- Save general registers, establish address-ability, etc., process parameter list to establish processor operating characteristics and execution limits. Store in Processor Operating characteristics Table (POT).

- Establish communication with HASP in order to obtain OS JOB-oriented statistics, i.e., lines printed, cards read/punched, etc.

- OPEN input data set (SYSIN) and output data set (SYSPRINT). Process input form HASP and scan for processor requests (??EXEC) or end-of-file. Flush and list on SYSPRINT with appropriate title, all input records until valid one found. All this may be performed in the FLUSH routine.

- A new batch is recognized when the accounting field on the '??EXEC' record changes. Old accounting is closed out on receipt of a physical end-of-file or a new accounting field.

- Link to ACCOUNT module Job Accounting Routine (JAR) with appropriate entry code to initialize job batch accounting, or close it out. An accounting record is built and subsequently written to the installation accounting data set for each batch account number. (File record formats appear in Appendices III, IV and VIII.

- Process the ??EXEC record to determine which processor is required.

- Determine if a processor was previously in operation. If so, see if it terminated normally or abnormally.

- If a processor was executing and went to normal completion, then accounting was properly closed out; another (or the same) processor may be attached as required by the ??EXEC record (or re-entered - see Appendix VI).

- If a processor was executing and ended abnormally, then accounting must be closed off for the current user (both batch accounting record (BAR) and group-account statistics (GAS)).

- If a new processor is required, then attach the required type as follows:

  - LOAD the processor and record the entry address in the MEAT.

  - LOAD then ATTACH (first time only) or re-enter a module linkage routine (MODLIN), and pass it a parameter list of the following form:

    A(MEAT)

    which is described in Figure 2.

  - The POT contains various operating characteristics required by the processor, such as maximum lines output, maximum execution time per problem, subscripting restrictions (PL/1), etc. The first half-word is on a half-word boundary and contains the length of the list following it. (See Appendix II.)

  - PEP is the processor entry point address which is obtained through the action of the LOAD macro whenever a new processor is required.

  - MOSS is the Multi-tasking Operations Synchronization and Switching routine which handles all scheduling of the monitor services required by the processor (and monitor), and re-scheduling of processor services.

  - JAR is the Job Accounting Routine described in Section 4.

  - CRAP (Cancel and Re-load A Processor) is entered from MODLIN after return has been made from a processor. This will only occur when MOSS has determined through FLUSH that a different processor is required.

  - SIT is discussed in Appendix VI.

  - MODLIN will pass control (CALL) to the processor via the entry point (PEP) address, and supply a parameter list consisting of the address of the MEAT received from the monitor via ATTACH.

The Processor. When a processor is initially given control (by the monitor raising its dispatching priority via the CHAP macro), it first processes the parameter list passed to it via MODLIN.

- Values are obtained from the POT to use as execution limits for problem programs.

- Addresses are obtained from the MEAT to be used in various linkages to the monitor, i.e., to the JAR for GAS at problem program initiation and termination; to the FLUSH routine when errors are encountered that cannot be handled by the processor; a Scheduling Information Table (SIT), used in the passing of control between Monitor and Processor.

- Procesor input is then scanned and used in the normal manner for batch-processing.

- At problem initiation, a call is made to the Job Accounting Routine (JAR) with the entry code set to 8, and the group account field from the problem header record which is used to look up a value in the Group Index Number (GIN) Table, which is then used as an index into the GAS table. The value is verified, group accounting is updated, checks made for excessive use of resources (if in effect) then return to the processor with an appropriate condition code, i.e.,

  0 = proceed normally

  4 = unverified group no.

  8 = resource limits exceeded

- At problem termination, a call is made again to the JAR with the entry code set to 12. At this point, the GAS table must be updated in regards to execution time, cards read, lines and/or pages printed, job count, successful completions. This may be done in one of three ways:

  - The processor accumulates these values in the GAS table, since they are not readily accessible to the JAR.

  - The JAR obtains whatever is available from HASP (via an SVC?), and the remainder is passed by the processor.

  - A combination of these two methods.

- Upon abnormal completion of the processor, control is given to the monitor ETXR, where appropriate action is taken.

- Upon encountering an end-of-file condition, instead of exiting immediately from the processor task, it may be more expedient to POST the monitor task and schedule a synchronous entry back into it, WAIT the procesor task, and thereby keep a dormant processor in main storage while the next logical functions are being carried out by the monitor, i.e., processing a request for the same processor. If not required, the dormant procesor may then be DETACHed. The two event control blocks (ECBs) labelled

MICE (Monitor Inter-Communication Element) and PRICE (Processor Inter-Communciation Element) have been supplied for such purposes. (Refer to Appendix VI for further discussion of this topic).

6. INCOMPLETE DEFINITIONS AND EXPLORATORY PROCEDURES

At various points throughout this document, values and symbols are arbitrarily assigned to processes and parameters, which require further qualification. An effort will be made at this point to provide explanations of the most pertinent of these:

- The FABLE input job stream, as suggested in Figure 1, is a concept of what will be expected as input from HASP using the Batch Scheduling facility. It is not indicative of the type of batch records which will be submitted at a HASP reader (except possibly in the initial testing stages of FABLE development). The intention is that the users of the facility submit a batch of one or more problems for processing by one or more batch processors, intermixed if desired. Each problem would contain the necessary header and control records normally required by the processor, and in addition, header card accounting peculiar to FABLE. The batch would be prefixed by the usual OS-HASP JOB record.

It is expected that the FABLE monitor, together with the WATFIV batch processor, but without the accounting facility, could be implemented in about two(2) programmer/months.

Processor and batch timing appear to be a nebulous process at present. It is anticipated that an SVC will have to be provided which will, upon request, return the value of CPU time remaining in the caller's job step. This will be used in the accounting module.

Some modifications will be required to HASP, particularly in the Batch Scheduling facility.

The ability of the monitor and the attached processors to collectively reference the SYSIN data set created by HASP, some by forward reference to DD-name in a procedure (e.g., WATFIV's FT05F001).

In an effort to minimize the modifications necessary to each batch processor attached by FABLE, it would be desirable to establish a set of standard interfaces, in the form of macro-instructions which could be inserted into the processor coding at the appropriate control points (i.e., module entry, problem initiation and termination). Of course, due to design differences in the various processors, some additional coding in each will be necessary.

7. SYSTEM SUMMARY AND RECOMMENDATIONS

From the foregoing discussion, it appears that, should one proceed to include in his partition monitor system all the desirable operating characteristics enumerated in the introduction (section 1) of this paper, considerable time and effort would have to be expended to achieve the goals and ideals presented in the remaining sections. Some of the work involved would entail experimentation with the design peculiarities of the host operating system. In addition, a well-organized approach is mandatory in regards to project formation, assignment of programming and analysis staff, team effort and timing estimates.

The FABLE system, as described herein, has been designed to be modular and open-ended. Therefore, it is recommended that any implementation of such a system be approached in the same manner, i.e.,

- Implement the most urgently required features first, and merely prepare the interfaces and linkages for other modules.

- Have more than one programmer working on various portions, or modules of the system at any one time. It should not be at all difficult to prepare dummy modules for use in testing others, which would provide a certain amount of programmer independence during the initial heavy development period.

- Macro-instructions and SVC's can be prepared in the preliminary stages.

- A rough estimate of the time required to fully implement FABLE (but not including the processors) would be four (4) programmer/months. An additional two (2) programmer/weeks might be required for each processor.

REFERENCES

1. HOTT Terminal Monitor - R. Vandem Ham, July 1971.

2. GY27-7236 IBM System/360 Operating System MFT Supervisor-Program Logic.

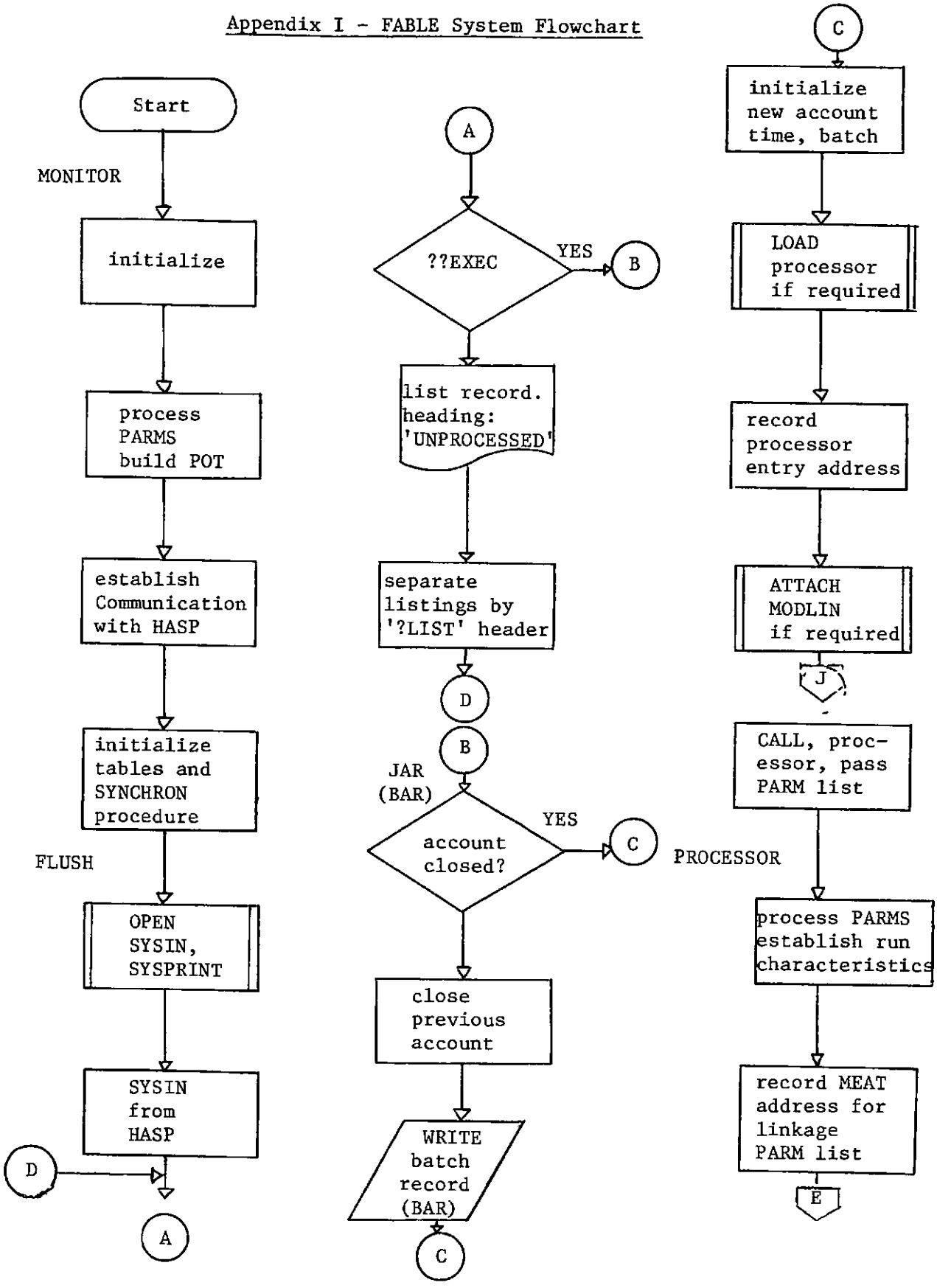
3. Houston Automatic Spooling Priority Program, Version III - Batch Scheduling Facility.

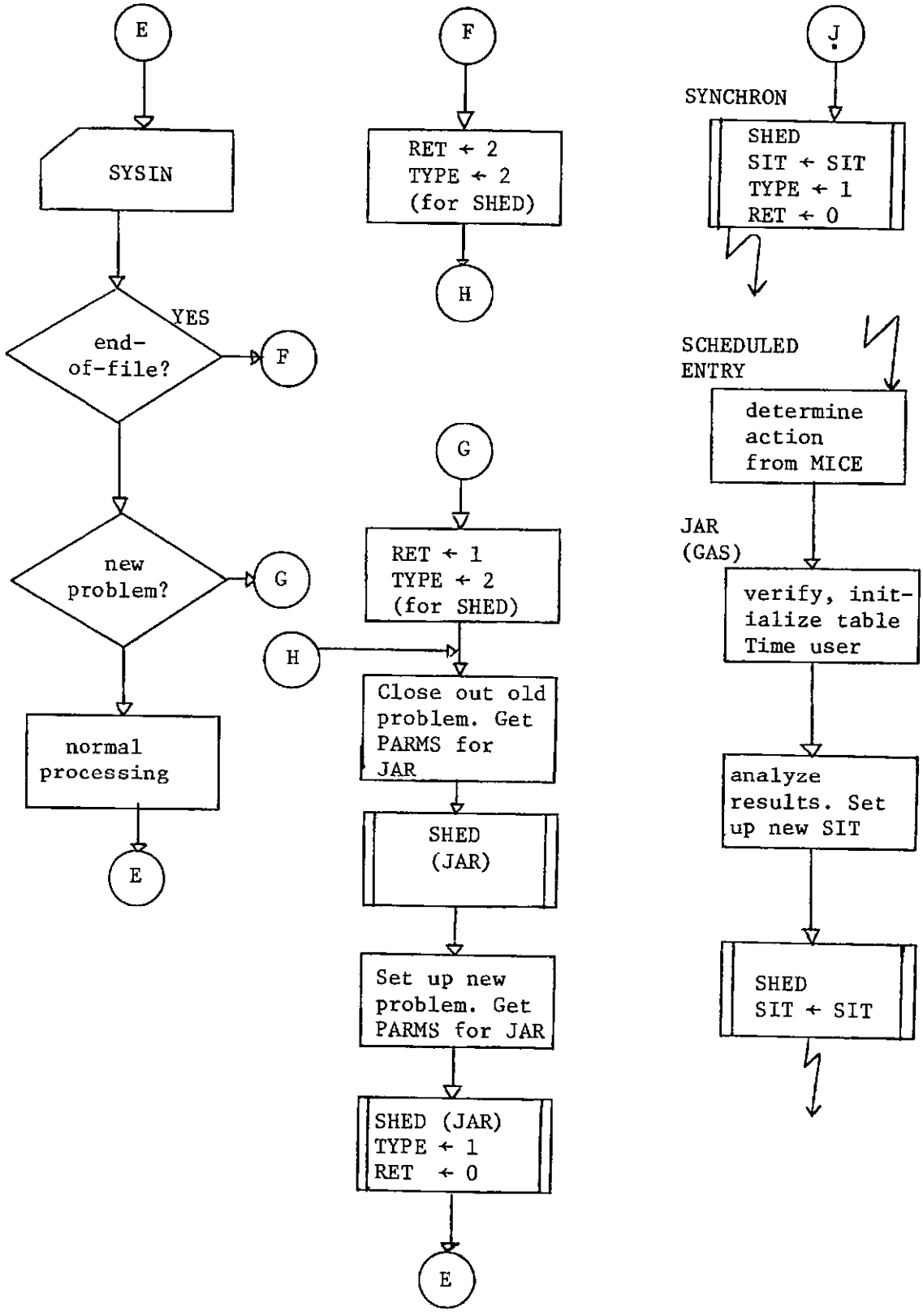
4. Communications of the ACM, Vol. 15, No. 3, Page 171.

5. Communications of the ACM, Vol. 15, No. 4, Page 221.

6. Baylor Executive System for Teleprocessing - Hobbs and mcBride, May 1971.

Appendix I - FABLE System Flowchart




E
SYSIN
end-
of-file?
YES
F
new
problem?
G
normal
processing
E
F
RET ← 2
TYPE ← 2
(for SHED)
H
G
RET ← 1
TYPE ← 2
(for SHED)
H
Close out old
problem. Get
PARMS for
JAR
SHED
(JAR)
Set up new
problem. Get
PARMS for JAR
SHED (JAR)
TYPE ← 1
RET ← 0
E
J
SYNCHRON
SHED
SIT ← SIT
TYPE ← 1
RET ← 0
SCHEDULED
ENTRY
determine
action
from MICE
JAR
(GAS)
verify, init-
ialize table
Time user
analyze
results. Set
up new SIT
SHED
SIT ← SIT

Appendix II - Processor Operating Characteristics

Table (POT)

(1)

0 POTLEN length of following list	2 POTMAXT maximum execution time (secs)	3 POTMAXP maximum pages allowed
4 POTMAXL maximum lines allowed	6 Reserved (for special limits imposed by some processors)	

NOTES: (1) This table is constructed within the FABLE monitor main storage area, as a result of processing the parameter list passed to FABLE from the control program, i.e., from the EXEC statement which scheduled FABLE for execution.

APPENDIX III - Group Accountint Statistics

(GAS) Table

TABLE 1

0 GASNUM group sub-account index number (from GIN table)	1 GASJOBS total jobs in this batch	2 GASTIME total batch execution time (maximum=655.35 secs)	
4 GASENDS total jobs in batch with normal ends	5 GASPAGES total printer pages used (in tens)	6 GASCARDS total cards read (in tens)	7 GASLINES total lines printed (in tens)

TABLE n

(same as Table 1)

NOTES: (1) Initially there will be a maximum of 30 such tables comprising a total of 240 bytes in the FABLE monitor main storage area.

(2) This 30-record buffer will be used to build a GAS table for each batch by account number.

(3) At the end of batch, the buffer will be collapsed to eliminate unused tables, and the resultant buffer written out to the accounting data set together with the batch accounting record (BAR).

(4) When each GAS table is copied to BAR, it will be prefixed by GINAME taken from the GIN table pointed to by GASNUM.

Appendix IV - Batch Accounting Record (BAR)

Physical Record Layout

Beginning of SMF logical record

-4 BARRDW Record descriptor word prefix	
BARLEN length of physical record	-2 Zeros

Beginning of FABLE logical record (SMF format)

0 Reserved (Zero)	1 BARTYPE SMF record ID:X'FE'	2 BARTMEND Time batch ended in hundredths of a second
4 BARTMEND Continued		6 BARDTEND Date batch ended in form 00YYDDDF
8 BARDTEND Continued		10 BARSYSID System identification
12 BARMODID Model identification		14 BARTMBEG Time batch started in hundredths of a second
16 BARTMBEG Continued		18 BARDTBEG Date batch started
20 BARDTBEG Continued		22 BARCARDS Total cards read in batch (maximum = 65,535)

Batch Accounting Record (BAR)

(Part 1 of 2)

Batch Accounting Record

(Part 2 of 2)

24 BARLINES Total lines printed in batch (maximum = 65,535)	26 BARPAGES Total pages used in batch (maximum = 65,535)
28 BARCODE Completion code when batch ended	30 BARCORE Maximum main storage used by batch.
32 BARIO Total I/O operations performed in batch (EXCP's)	34 BARNAME Programmer name from FABLE EXEC record (max = 20 bytes)
52	54 BARCPUT CPU time used in batch (maximum = 655.35 secs)
56 BARCNTI length of next field	57 BARACCT Accounting field from FABLE EXEC (or JOB) record (variable length)
BARCNT2 length of next field	(57 + BARCNT1) BARPUS Processor utilization statistics (see Appendix VIII)
BARCNT3 length of remaining record (GAS)	(57 + BARCNT1 + BARCNT2) BARGAS Condensed GAS table recorded during batch (variable length) (see Appendix III)

Appendix V - Statistical Accounting Limit Table (SALT)

0 Reserved	1 SALTIME Total CPU time remaining (maximum = 167,772.15 secs)
4 Reserved	5 SALLINE Total printer lines remaining (maximum = 16,777,215)
8 SALPAGE Total printer pages remaining (maximum of 65,535)	10 Reserved

NOTES: (1) One such table exists for each group index number, which comprises a block associated with each account number, and resides on direct access storage.

(2) The table is read into main storage by the FABLE monitor accounting routine at batch initiation time for verification, and written back at batch termination for updating.

(3) This scheme, among others associated with accounting, may not be implemented until later.

APPENDIX VI - INTER-TASK COMMUNICATION AND PROCESS SYNCHRONIZATION

Synchronization of logical processes and events which are effected within one task, in the performance of a service scheduled by another task, has been the subject of some discussion in the computing literature of today [4,5], and has, to some extent, been used in one form or another in various time-sharing and other multi-tasking systems [6].

The heart of the problem, as it applies, for example, to OS/360 MFT with sub-tasking, may be described as follows:

- Task B is a sub-task of task A, and is processing rather independently of the MOTHER task (A), except that at various points in its operation it requires pre-defined services to be performed for it by the MOTHER, or it is required to 'check-in' with MOTHER at scheduled times. These are all considered synchronous entries into the MOTHER task.

- One way to carry out these synchronous transfers is to simply set up the necessary linkages with the required entry points in the MOTHER by employing pointers which were passed to task B by the MOTHER when task B (DAUGHTER) was ATTACHed.

- When a transfer is accomplished, the required code in the MOTHER is executed under control of DAUGHTER and a return is made back to a DAUGHTER process.

However, timing statistics and other accounting are left in a rather questionable state using this method of processing out-of-task services. In addition, the MOTHER task, which is most likely performing the duty of a monitor, has no stringent control over the process being carried out, except that should an abnormal termination occur, MOTHER will receive control again through an end-of-task exit routine (ETXR), which is an asynchronous entry. This is not succinct enough for the objectives outlined in this document; therefore, a more precise definition will be attempted.

To confine this discussion once again to two tasks only, let us define two inter-communication elements (ICE), which shall at this time be OS/360 event control blocks (ECB). We assign them the titles Monitor Inter-Communication Element (MICE) and Processor Inter-Communication Element (PRICE). They may now be used in the following manner:

- When MOTHER attaches DAUGHTER, it will relinguish control to it in the following manner:
  - MOTHER will upgrade DAUGHTER's dispatching priority, and then WAIT on MICE.

- When DAUGHTER requires MOTHER's assistance, it will schedule such an event as follows:

  • DAUGHTER will POST MICE and also indicate what service routine is to be given control, by using MICE.

  • DAUGHTER will WAIT on PRICE

- When MOTHER receives control once again, it determines, using MICE, what service routine to enter, and will:

  • POST PRICE with 'process start' flag set, and link to the routine.

  • DAUGHTER will determine from PRICE whether to carry on or to WAIT again on PRICE.

  • MOTHER will return from the service routine and POST PRICE once again with appropriate completion code.

  • MOTHER will WAIT on MICE once again.

The processes described above should have the desired effect upon task and process timing, and also elicit an autonomy of control. In addition, it would be desirable to go one step further, and prepare a set of macro-instructions which could be inserted into any processor, and with the proper manipulations of global set symbols, they would minimize the source code modifications required whenever a new processor is to be adopted for use with the monitor. Such macros may take the following form:

SHED - Schedule Higher-priority Event Dynamically

[label]	SHED	[symbol / value / 0] [,SIT= {address / 0}] [,ENTRY= {symbol / value / 0}] [,SIG= START / END] [,WAIT= {YES / NO}] [,RESUME= {address / 0}] [,RET= {HOLD / RLSE}]

name
is a symbolic offset established through the SMELL macro, to the MEAT. If this operand is omitted, 0 is assumed, which is a status request only to the MOSS routine.

SIT
is the address of a user-specified Scheduling Information Table which will be required for multi-processing with more than one sub-task. If this operand is omitted, 0 is assumed, which causes the standard SIT supplied by the monitor task to be used.

ENTRY
is a symbolic entrance code to be passed to the scheduled routine whose address (from the MEAT) was specified by the name operand. If omitted, a code of 0 will be passed.

value
is an absolute offset to the MEAT or an entry point obtained from it.

WAIT=YES
specifies to the MOSS, that the scheduler is not to be re-activated until the scheduled event is completed. If this operand is omitted, YES is assumed.

WAIT=NO
specifies that the scheduler is to be re-activated immediately.

RESUME
is the address of the location in the scheduler which is to be given control when the scheduler has been re-activated. If omitted, 0 is assumed which causes control to be given to the instruction following the SHED macro.

RET=HOLD
indicates to the MOSS routine that on completion of the scheduled event, the scheduler is not to be re-activated. In such a case, the MOSS routine must determine the next course of action, rather than de-activating itself.

RET=RLSE
indicates to the MOSS routine that upon completion of the scheduled event, the scheduler must be re-activated. If operand is omitted, RLSE is assumed.

SIG=START
indicates to the MOSS routine that immediately the requested event is scheduled, the scheduler must be notified via the SIT (bit 3 of SITRET)

SIG=END
indicates to the MOSS routine that the scheduler must be notified of the completion of the scheduled event via the SIT (bit 7 of SITRET). If this operand is omitted, END is assumed.

0 SITCODE entry code to pass to scheduled routine	1 SITRAD address of higher-priority routine to be scheduled.
4 SITTYPE (1)	5 SITMICE address of Monitor Inter-Communication Element (MICE)
8 SITRET (2)	9 SITPRICE address of Processor Inter-Communication Element (PRICE)

NOTES: (1) SITTYPE - type of scheduling

```
xxxx....  reserved
....1...  retain control.  Do not POST scheduler
.....1..  signal start of scheduled event
.....0..  signal end of scheduled event
......1.  POST scheduler immediately
......0.  POST scheduler when event completed
.......1  initiate scheduled process
.......0  status request
```

(2) SITRET - action on return to scheduler

```
xxx.....  reserved
...1....  event has been scheduled
....1...  terminate scheduler process
....0...  continue scheduler at resume address
.....1..  POST delayed indefinitely by scheduled process
......1.  POST delayed indefinitely by scheduler process
.......1  scheduled event failed
.......0  scheduled event completed successfully
```

Scheduling Information Table

SMELL - Specify Multi-processing Entry

List Locations

[label]	SMELL	MEAT = (name1,name2,,,,)

MEAT
is the Multi-tasking Entry Address Table whose address is passed to the program issuing the macro via GPR 1. Offsets, in steps of 4 beginning with zero, within this table are to be assigned symbolic names, in order, from the sublist.

name1,name2...
is a sublist of symbolic entry point names to be assigned to the offsets in the MEAT, one for one, in the order specified. These symbols may subsequently be used in the SHED macro for specifying entry points of processes to be scheduled.

APPENDIX VII - Group Index Name Table (GIN)

0 GINNUM index number	1 GINNAME 8-character course no., dept. no., or other identification for intra-group user
9	10 (This entry and all following are identical to 0-8)
(9*n-9)	

NOTES: (1) One such table will exist on direct access storage for each group, and will be specified by the group.

(2) There exists a one-to-one correspondence between each entry in the GIN and GAS tables.

(3) Used by JAR to look up GAS index number from 8-character identifier found on processor header record.

APPENDIX VIII - Processor Utilization Statistics

(PUS) TABLE

TABLE 1

0 PUSPROC Processor Program Module Name			
4 PUSPROC			
8 PUSNUM index or ID number for processor	9 PUSJOBS total jobs processed by processor	10 PUSTIME total execution time used by processor	
12 PUSENDS total jobs processed with normal endings	13 PUSPAGES total printer pages used by processor (in tens)	14 PUSCARDS total cards read by processor (tens)	15 PUSLINES total lines printed by processor (tens)

TABLE n

(n*16-16) (Same as Table 1)

NOTES: (1) There will be one such table created within the FABLE monitor for each processor which is attached.

(2) The resultant tables will be written onto the accounting data set as part of the BAR at the end of each batch.