RJ2450(32255)1/15/79
Computer Science

J. Gray

Research Report

A SHARED SEGMENT AND INTER-PROCESS COMMUNICATION FACILITY FOR VM/370

J. N. Gray
V. Watson

IBM Research Laboratory
San Jose, California 95193

This report is a revised version of RJ 1579.

IBM Research Division
Yorktown Heights, New York · San Jose, California · Zurich, Switzerland

Copies may be requested from:
IBM Thomas J. Watson Research Center
Post Office Box 218
Yorktown Heights, New York 10598

RJ2450(32255)1/15/79
Computer Science

A SHARED SEGMENT AND INTER-PROCESS COMMUNICATION FACILITY FOR VM/370

J. N. Gray
V. Watson

IBM Research Laboratory
San Jose, California 95193

ABSTRACT: VM/370 has been modified to allow memory sharing and signalling among several virtual machies. This document describes the design, principles of operation and implementation of these changes and also serves as a user's manual for the enhanced system. Memory sharing is on a segment basis. Inter-process communication is based on the multiprocessing instructions of System 370.

This report is a revised version of RJ 1579.

TABLE OF CONTENTS

0. OVERVIEW

VM/370 has been modified to support an experimental multi-user data management system [Astrahan]. In particular the following facilities were added:
* Dynamic sharing of primary memory among virtual machines.
* Inter-process communication among virtual machines.

VM/370 allows machines to dynamically share disks. However, VM was modified to allow users to concurrently access common segments of primary memory. This avoids the update problems inherent in multiple copies of shared disk data being mapped into different primary memories. Read-Write shared segments allow all users to see common data in primary memory and locking may have the granularity of a byte rather than of a disk page or larger physical unit.

Since the machines share data, they must synchronize their reads and updates via some form of inter-process communication. Machines must be able to SIGNAL and WAIT for one another. The use of these primitives to produce a transaction structure, locking protocol and recovery mechanism is documented in [Astrahan].

The problem of authorizing access to these facilities is handled as follows: Two new privileges, DBSHARE privilege and MONITOR privilege were added to the system. A process must have DBSHARE privilege in order to send or receive a signal via the inter-process communication facility, create or destroy a shared segment owned by that process, link a shared segment or exercise the commands which query the status of shared segments. Further, shared segments are protected by passwords. One must know the password to link a segment unless he is the owner (creator) of the segment. MONITOR privilege allows one machine to create, destroy, link and detach shared segments to and from other machines. Both DBSHARE privilege and MONITOR privilege are controlled by the VM administrator when he defines machines in the system directory.

In proposing changes to VM/370 we have been conservative. Wherever possible we have adopted an existing definition (e.g. Signal Processor instruction for inter-process communication) rather than inventing a new feature. Where no acceptable facility exists we have proceeded by analogy with 'similar' mechanisms found in VM/370. In particular, segment sharing is designed in analogy to minidisk sharing in VM/370.

The changes described here became part of the "floor" VM system at San Jose in 1975 and have been in use at several sites since then. It runs with Releases 3, 4, and 5 on VM.

Disclaimer: We wish to emphasize that the VM/370 modifications described in this report have been developed for our experimental environment, and are not generally available.

I.1 Principles of Operation

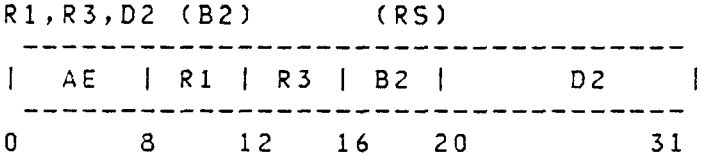
As mentioned above, the inter-process communication that we have added is based on the processor signalling and response feature of the multiprocessing facility provided by the IBM System/370. We also implemented some extensions to the Signal Processor instruction and the external interrupt architecture. The sections dealing with the architecture are patterned after the IBM System/370 Principles of Operation [System 370]. The reader is assumed to have an understanding of System/370 as well as some knowledge of the internals of VM/370 [VM/370].

I.1.1 The Signal Processor Instruction

The 370 processor Signal Processor instruction facilitates communication among processors in a multiprocessor system. It provides for transmitting and receiving the signal, decoding a set of assigned order codes, performing the specified operation and responding to the signalling processor.

SIGNAL PROCESSOR

SIGP R1,R3,D2 (B2) (RS)



An eight-bit order code in the second operand and a parameter in the first operand are transmitted to the machine designated by the processor address contained in the third operand. The result is indicated by the condition code and may be detailed by status assembled in the first operand.

The address of the register containing the 32-bit parameter is odd and is equal to R1 or one larger.

The second operand contains an eight-bit order code in bits 24-31. Bits 8-23 of the second-operand are ignored.

The register designated by the third operand contains a 16 bit binary number which forms the processor address; High order bits are ignored.

The resulting Condition Code:

0	Order code accepted
1	Receiver status stored
2	Receiver is busy
3	Receiver is not operational or privilege exception

On execution of the instruction, if the condition code is zero, no status is returned and the register designated by the first operand remains unchanged. If condition code of one was set, status information is generated and returned in the register designated by the R1 field.

I. INTER-PROCESS COMMUNICATION FACILITY

If two or more processes share updatable storage they need some mechanism to solve the readers and writers problem (to synchronize their reads and updates so that no inconsistent data is read and no updates are lost). The 370 provides atomic instructions (Test and Set, Compare and Swap) which allow the implementation of semaphores and queues [System 370, Appendix I]. Using these instructions, two or more processes can synchronize with one another.

If one process finds a semaphore set or a queue non-empty then the process may have to wait. The forms of waiting can be dichotomized as:

Busy wait: in which the waiting process continually polls the queue seeing if its turn has come.

Interrupt wait: in which the waiting process is suspended and waits for a signal that it can proceed.

We have opted for an interrupt wait mechanism because it uses less processor time and because it allows a fair resource scheduler. We wanted a SIGNAL/WAIT mechanism which is both simple and fast (less than 5000 instructions to send and receive a signal). Given these criteria, we decided to add simulation of the 370 Signal Processor Instruction to VM/370 [System 370].

SIGP is a System 370-multiprocessor option instruction. It allows one processor to send various orders to another, including an external interrupt. This provides a signalling facility. By loading a PSW with external interrupts enabled and the wait bit set a second processor can wait for such a signal. Preliminary tests indicate that such an exchange requires 6000 370 instructions.

The Signal Processor (SIGP) instruction has many bells and whistles for inter-process communication. It has things like Start, Stop, External Call, as well as more esoteric order codes like Initial Microprogram Load. It is designed to allow one processor to control another. This is a rather privileged operation and so in our VM/370 implementation it is authorized as follows: To be able to communicate at all both machines (the sender and receiver) must have DBSHARE privilege. If the order code is 'sensitive' the sender must also have MONITOR privilege. External calls are not sensitive, but all other orders are.

VM has been modified to assign a unique processor address (two bytes) to each virtual machine at login time so that SIGP's can be addressed to particular machines. Thereafter, the processor can discover its address by executing the Store Processor Address instruction (STAP), a standard 370 instruction which we have added to VM/370's repertoire of simulated privileged instructions.

An extension to the QUERY command (QUERY CPUS) displays the processor address of each user logged into the system.

I.1.1.1 Function Definition

System 370 describes twelve orders for communication among processors. The orders are specified in bit positions 24-31 of the second operand address of the Signal Processor instruction.

We have implemented an additional order:
External Call with Parameter - Code 0D.

As mentioned above, the sender and receiver must have DBSHARE privilege. If the type of order is sensitive, the sender must have MONITOR privilege (see table below). If a violation is detected condition code 3 is set .

The orders are encoded as follows:

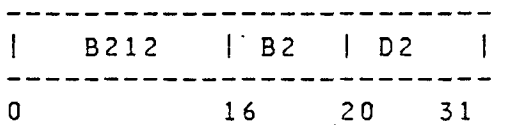
code	order	privilege
00	Unassigned	
01	Sense	S
02	External Call	S
03	Emergency Signal	S
04	Start	M
05	Stop	M
06	Restart	M
07	Initial Program Reset	*
08	Program Reset	M
09	Stop and Store Status	M
0A	Initial Microprogram Load	*
0B	Initial processor reset	*
0C	processor reset	*
0D	External Call w/Parameter	S
S	DBSHARE	
M	MONITOR	
*	Not supported	

Appendix A describes in detail the Signal Processor orders. The external-interruption conditions generated by the Emergency Signal, External Call and the Extended External Call (with parameter) are described in Appendix B. Appendix C details the status information generated by the Signal Processor instruction.

I.1.2 The Store Processor Adress Instruction

Each virtual machine is assigned a unique two-byte address at logon. A processor is designated by specifying this address in the processor address field of a Signal Processor instruction. The processor signalling an emergency signal or external call is identified by storing this address in the processor address field with the interruption. A program can determine the address of the processor by means of the instruction Store Processor Address.

STAP D2(B2) (S)



The address (2 bytes) by which this processor is identified (in a multiprocessing system) is stored at the halfword location designated by the second operand address.

I.2 Implementation

I.2.1 Relation to Previous Work

In implementing the inter-process communication in VM/370 we were very fortunate to obtain a version of Virtual Multiprocessor support for VM/370. The Virtual Multiprocessor system was intended to support VS/2 Rel.2 development. The System was developed in Poughkeepsie by Tom Gilbert (the original code for the SIGP was written by Peter Sih in San Jose).

Out of this system we incorporated into our system the code dealing with the privileged instruction simulation support for Signal Processor and Store Processor Address and added code to support the Extended External Call. We could not use any of the other changes as they dealt primarily with support of multiprocessors and had been written for Release 1 of VM.

I.2.2 Outline of Implementation

We have added a new module to VM/370 (DMKAES) which contains the the Signal Processor simulation. The directory program has been modified to recognize two new CP command privilege levels to safeguard the use of inter-process communication (and segment sharing). We also added code which assigns processor addresses to users logging into the system for use in the SIGP instruction. Finally, we added the new CP console command to display processor addresses of all users.

For a detailed description of changes to VM/370 see Appendix D.

II. DATA SHARING

II.1 Existing Facilities in VM/370

VM/370 has a facility for multiple users to read-share common segments of primary memory. This Named System facility is only intended for the sharing of re-entrant code among users so that there is at most one copy of the code in primary memory. The named system facility does not allow read-write shared access to primary memory (in fact much of the complexity of the design is necessary to prevent users from updating the shared system). Hence it is not appropriate for our needs.

Release 3 of VM added support of discontiguous saved segments. Although they provide more flexibility - you can attach and detach segments to and from your virtual machine - they do not have read-write access and must be named and saved in much the same way as the Named Systems.

II.2. Sharing Primary Memory

Our goal was to allow multiple virtual machines to concurrently read and write common primary memory areas and yet allow them to also have private memory.

II.2.1 The Model of Memory Sharing

II.2.1.1 The Quantum of Sharing

First we argue that a page table is the appropriate unit of sharing. We proceed by analogy to the hardware and to VM/370: the unit should correspond to a logical model of the physical resource. The 370 Dynamic Address Translation hardware depends directly on the Segment Table and the Page Table and of course on page frames. Thus the candidates for the quantum of sharing are page, page table (= segment) or segment table (= address space). The smallest unit of sharing would be a 4K page. Sharing such a small quantum would require substantial bookkeeping on each page frame to record its global name, password and for each shared page, its current location. Also there are some obscure technical problems with having multiple page and swap table entries describing the same page. Since we expect to be sharing aggregates of pages we have decided against a page in favor of the larger quantum of a page table. Using page table sharing, one can simulate the sharing of segment tables among machines by giving each machine a separate copy of the same segment table. (There is a problem about giving each machine the same page zero.) The converse is not true, however; if one could only share segment tables then two machines would not be able to share memory and have some private memory. For these reasons we choose to have page tables as the smallest unit of sharing.

II.2.1.2 The Sharing Mechanism

In keeping with the spirit of VM/370, shared segments are treated as logical devices. Each shared segment has an owner, a name and a password. A shared segment is a volatile storage medium which may be read and written by several machines but which disappears at each system shutdown or crash. Access to the segment is controlled by the password. This is in analogy to the way minidisks are shared/protected among users in VM/370.

The CP component of the system is not responsible for the preservation of these segments. That is, both the contents and the definition of a shared segment are volatile. Checkpoint and recovery of the contents of the shared segments are the responsibility of the user machines and each segment's owner.

All shared segments are 64K bytes long. A shared segment is created when a user defines one of his private segments to be shared giving it a name and a password (SEGMENT DEFINE). The initial contents and keys of the segment are inherited from this private segment in the owner's address space. After a segment is defined, any machine with the password may link the segment (SEGMENT LINK), thereby binding the segment to its virtual memory at a specified segment slot. A machine can implicitly or explicitly execute a SEGMENT DETACH command which replaces the shared segment with a private segment. DEFINE STORAGE, SYSTEM RESET, SYSTEM CLEAR, LOGOUT and IPL implicitly detach all shared segments from a machine. SEGMENT RELEASE is identical to SEGMENT DETACH (it is included for symmetry). Segments are deallocated when they are no longer linked by any machine.

When a machine is created, it has a memory consisting of a number of private segments. It may vary the number of segment slots it has by a DEFINE STORAGE command. Thereafter, the machine can link any segment (for which it has the password) to any slot of its segment table (except slot zero). Once a machine has linked to a segment, the machine has read-write access to all bytes of the segment and its storage keys. It may synchronize with other machines by using the mechanisms provided by the inter-process communication facility.

Thus we have described:

- The unit of sharing: a page table = 64K bytes.
- The authorization scheme: passwords.
- The definition and binding mechanism: SEGMENT DEFINE, LINK, DETACH and RELEASE.

II.3 Principles of Operation

II.3.0 General Information

The Segment Sharing facility is handled through a new CP command. In order to use the segment sharing command, a user must have D privilege, i.e. the DBSHARE option in his/her machine description in the directory.

Displaying shared segment definitions -

Issuing the 'QUERY SHARES' command will list all shared segments in the system.

Displaying links to shared segments -

'QUERY SEGLinks segname' will display all 'userids' linked to a specified segment.

Displaying memory -

'QUERY MEMORY' command will display the contents of the user's segment table. If the memory contains shared segments, the name of the shared segment and the owner are displayed. Non-shared segments are displayed marked as 'private' or 'invalid'.

Defining a segment -

A user may define any segment of his/her current virtual memory as shared, with the exception of segment zero. Once a segment is defined it remains part of the system until the last user linked to the segment logs off.

Deleting (releasing) a shared segment definition -

An owner may delete shared segment definitions from the system provided no one else is linked them.

Deleting shared segment definitions also releases the shared page and swap tables, replacing them with a private segment.

Linking a shared segment -

In order to link to any shared segment in the system, the user must know the password. Once a segment is linked, it becomes part of the user's virtual memory space with read and write access. Linking can only be done in the user's current memory. This means that a shared segment would necessarily overlay some existing segment.

Detaching a segment -

On detaching, shared segments are replaced by new, initially empty private segments. If the use count of a segment being detached goes to zero, the segment definition (SHRTABLE) is also released.

Defining a Shared Segment

SEGment DEFine

Privilege - DBSHARE

The SEGment DEFine command allows a user to declare a segment of his virtual storage as shared.

Segment names are unique across all machines, and only one shared segment can be defined for a given page table.

The format of the command is:

```
SEGment DEFine |userid| XXX <AS> segname password (NOMSGE)
               |*     |
```

where:

userid — is the user identification of the virtual machine in which the shared segment is being defined. (A machine has to have MONITOR privilege in order to issue SEG DEFines for segments in other machines.) The user denoted by userid must be logged on.

* — self

XXX — is the segment number (in hexadecimal) which is to be defined as shared.

segname — is the one to six character name which is to be assigned to the segment. The keyword <AS> may be omitted.

password — is a one to eight character string which is to be the password to linking the segment.

NOMSGE — suppresses the typing out of responses.

Responses:

0 - Segment defined
1 - Invalid parameter
2 - User not on system
4 - Invalid segment number
5 - Privilege exception
6 - Invalid password
7 - Invalid segment name
8 - Duplicate segment name
11 - Segment already shared

Releasing a Shared Segment Definition

SEGment RELease

Privilege - DBSHARE

The SEG RELease command is detaches the designated segment from the designated user and deletes the definition of the segment if it is not by others. The "hole" left by the detach is replaced by a private segment. The 'ALL' option permits the release of all the shared segments defined and presently linked to the user. If there are any other links to the segment the definition is not deleted. SEGMENT RELEASE is identical to SEGMENT DETACH.

The format of the SEGment RELease command is as follows:

```
SEGment RELease |userid| |segname| (NOMSGE)
                |*     | |ALL    |
```

where:

userid — is the user identification of the VM issuing the command and must be the owner of the segment unless the issuing machine has Privilege M.

* — self

segname — is the name of the segment to be released from the system.

ALL — indicates that all of the shared segments in user's memory are to be released from the system. The user issuing the command must be the owner of the segments or have Privilege M.

NOMSGE — indicates that typing out of responses is to be suppressed.

Responses:

0 - Segment definition released
1 - Invalid parameter
2 - User not logged on
3 - Segment not defined
5 - Privilege exception
7 - Invalid segment name
10 - Segment not released
12 - Segment not linked

Linking a Shared Segment

SEGment LINK

Privilege - DBSHARE

The SEGment LINK command allows a user to replace a segment of his virtual storage with a named shared segment.

The format of SEGment LINK command is as follows:

```
SEGment LINK |userid| segname <AS> XXX password (NOMSGE)
             |*     |
```

where:

userid — is the user identification of the virtual machine issuing the command (unless the issuing machine has Privilege M). The user denoted by userid must be logged on.

* — self

segname — is the name of the shared segment to be linked. The segment must be unique in the user's virtual storage.

XXX — is the number of the segment (in hexadecimal) to be replaced. The segment must be valid.

password — is a one to eight character string that must match the segment password in the shared segment definition.

NOMSGE — indicates that no responses should be typed out.

Responses:

0 - Segment linked
1 - Invalid parameter
2 - User not logged on
3 - Segment not defined
4 - Invalid segment number
5 - Privilege exception
6 - Invalid password
7 - Invalid segment name
9 - Segment already linked
11 - Segment already shared

Detaching a Shared Segment

SEGment DETach

Privilege - DBSHARE

The SEGment DETach command allows a user to detach a shared segment and replace it with a private segment.

The format of SEGment DETach command is as follows:

```
SEGment DETach |userid| |segname| (NOMSGE)
               |*     | |ALL    |
```

where:

userid — is the identification of the virtual machine issuing the command (unless the issuing machine has Privilege M). The user denoted by userid must be logged on.

* — self

segname — is the name of the shared segment to be detached.

ALL — indicates that all of the shared segments linked to the user are to be detached.

NOMSGE — indicates that typing out of responses is to be suppressed.

Responses:

0 - Segment detached
1 - Invalid parameter
2 - User not logged on
3 - Segment not defined
5 - Privilege exception
7 - Invalid segment name
12 - Segment not linked

Query Memory, Query Shared Segments, Query Cpuids and Query Segment Links

QUERY

Privilege - DBSHARE

Four new options have been added to the QUERY command. The format of the command is as follows:

Query	MEMORY
	SHARES
	CPUS
	SEGLinks segname

The options for the added functions of the QUERY command are:

MEMORY displays the contents of the user's memory.

Response: XXX NAME OWNER
. . .
. . .

where:

XXX is the segment number

NAME is the segment name if it is a shared segment, otherwise marked 'private'. It may also be an 'invalid' segment or a segment moved from the fast paging device - 'migrate'.

OWNER is the name of the owner of the shared segment; blank if 'private'.

SHARES displays all the shared segments in the system.

Response: name owner
. .
. .

CPUS displays the processor addresses of all the users with DBSHARE or MONITOR privilege logged into the system.

Response: name processor address
. .
. .

Error Messages: Shared segments not defined

SEGLinks displays all userids linked to a shared segment 'segname'.

Response: userid
.
.

Error Messages: segment not defined

II.4 Implementation

II.4.2 Outline of Implementation

The code for the segment sharing commands and related routines is all contained in a new module, DMKSEG.

When a segment is declared as shared, a new shared segment definition block (SHRTABLE) block is created and the page table is flagged 'shared'. We have extented the SHRTABLE which is used for both the Named Systems and Discontiguous Shared Segments, to include the name of owner and password.

We have also added two new command privilege levels to control access to the segment sharing commands as well as inter-process communication. Unfortunately, existing privilege classes in VM could not be extended without considerable modifications. We therefore added the privilege in the form of an option (like ECMODE and REALTIMER).

For detailed description of the changes to VM see Appendix E.

III. REFERENCES

[1] Alex Chandra, Shirley Hsieh, 'SPY Users Guide', IBM Research memorandum, IBM Research Laboratory, Yorktown Heights, New York, 1974.

[2] Shirley Hsieh, Inter-virtual machine communication under VM/370', IBM Research Report No. RC 5147, IBM Research Laboratory, Yorktown Heights, New York, 1974.

[3] Anonymous. 'IBM System/370 Principles of Operation', Order No. GC22-7000-3, IBM Armonk, New York, 1974.

[4] Anonymous. 'IBM Virtual Machine Facility/370: Command Language Guide for General Users', Order No. GC20-1804-2, IBM, Armonk, New York, 1974.

[5] Lyn Wheeler, 'VM/370 extended virtual memory management', unpublished memo, IBM Cambridge Scientific Center, Cambridge, Mass., 1974.

APPENDIX A

Signal Processor Orders

The SIGP orders are defined as follows:

Sense (Code 01): The addressed CPU presents its status to the issuing CPU. No other action is caused at the addressed CPU. The status, if not all zeros, is stored in the general register designated by the R1 field, and condition code 1 is set; if all status bits are zero, condition code 0 is set.

External Call (Code 02): An External Call external-interruption condition becomes pending during the execution of the Signal Processor instruction. The associated interruption occurs when the CPU is interruptable for that condition and does not necessarily occur during the execution of the Signal Processor instruction. The address of the CPU sending the signal is provided with the interruption code when the interruption occurs. Only one External Call condition can be kept pending in a CPU at a time.

Emergency Signal CPU does not necessarily enter the operating state during the e is generated at the addressed CPU. The interruption condition becomes pending during the execution of the SIGP instruction. The associated interruption occurs when the CPU is not interruptable for that condition and does not necessarily occur during the execution of the SIGP instruction. The address of the CPU sending the signal is provided with the interruption code when the interruption occurs. At any one time the receiving CPU can keep pending one Emergency Signal condition for each CPU of the multiprocessing system, including the receiving CPU itself.

Start (Code 04): the addressed CPU is placed in the operating state. The CPU does not necessarily enter the operating state during the execution of the Signal Processor instruction. No action is caused at the addressed CPU if that CPU is in the operating state when the order code is accepted.

Stop (Code 05): The addressed CPU performs the stop function. The CPU does not necessarily enter the stopped state during the execution of the SIGP instruction. No action is caused at the addressed CPU if that CPU is in the stopped state when the order code is accepted.

Restart (Code 06): The addressed CPU performs the restart function. The

CPU does not necessarily perform the function during the execution of the SIGP instruction.

(The restart interruption provides a means for the operator or another CPU to invoke the execution of a program. The CPU cannot be disabled for this interruption.

A restart interruption causes the old PSW to be stored at main-storage location 8 and a new PSW to be fetched from location 0.

The restart interruption is initiated by activating the restart key on the system console. In a multiprocessing system, the operation can also be initiated at the addressed CPU by issuing Signal Processor specifying the Restart order.)

Initial Program Reset (Code 07): The addressed CPU performs initial program reset. The execution of the reset does not affect other CPUs and does not affect channels not configured to the CPU being reset. The reset operation is not necessarily completed during the execution of the Signal Processor instruction.

Program Reset (Code 08): The addressed CPU performs program reset. (Currently not supported.)

Stop & Store Status (Code 90): The addressed CPU performs the stop function, followed by the store-status function. The store-status operation consists in placing the contents of the current PSW and the program-addressable registers in permanently assigned locations within the first 512 bytes of main storage. The CPU does not necessarily complete the operation, or even enter the stopped state during the execution of the SIGP instruction.

Initial Microprogram Load (Code 0A): The addressed CPU performs initial program reset and then initiates the initial-microprogram-load function. (Not supported.)

Initial CPU Reset (Code 0B): The addressed CPU performs initial CPU reset. (Not supported.)

CPU Reset (Code 0C): The addressed CPU performs CPU reset. (Not supported.)

Extended External Call (Code 0D): An Extended External Call external-interruption condition is generated at the addressed CPU. The interruption condition becomes pending during the execution of the Signal Processor instruction. The associated interruption occurs when the CPU is interruptable for that condition and does not necessarily occur during the execution of the Signal Processor instruction. The address of the CPU sending the signal and a parameter are provided with the interruption code when the interruption occurs. In our implementation, there is no limit on the number of extended-external call conditions that can be kept pending in the receiving CPU.

APPENDIX B

Extended External Interrupt Structure

The external interruption provides a means by which the CPU can respond to various signals originating either from within or from outside of the system.

In BC mode the external interruption code is stored in bit position 16-31 of the External Old PSW. In EC mode the interrupt code is stored in locations 134-135.

Bits 6 and 7 of the interruption code, when one, indicate that additional fields have been stored as part of the interruption action. Bit 6, when one, indicates that the address of the processor issuing the interruption has been stored in locations 132-133. When bit 7 is one, it indicates that a parameter has been stored in locations 128-131.

An external interruption for a particular source can occur only when the CPU is enabled for an interruption by that source. PSW bit 7 and external submask bits in Control Register 0 have to be on for an interruption to be caused. (The use of submask bits does not depend on whether the CPU is in the BC or EC mode.)

External Interruptions generated by SIGP instructions

On issuing a SIGP instruction three external interruptions may be generated in the receiving CPU: the emergency signal, the external call, and the extended external call (external call with parameter).

The external interruption classes may be extended by increasing the number of distinct interruption conditions that can be identified.

Emergency Signal

An interruption request for emergency signal is generated when the CPU accepts the Emergency Signal order specified by a Signal Processor instruction addressing this CPU. The instruction may have been executed by this CPU or by another CPU with S privilege. The request is preserved and remains pending in the receiving CPU until it is cleared. The pending request is cleared when it causes an interruption and by CPU reset.

Facilities are provided for holding a separate emergency-signal request pending in the receiving CPU for each configured CPU, including the receiving CPU itself.

The condition is indicated by an external interruption code of 1201 (hex). The processor address of the CPU that issued the Signal Processor instruction is stored at locations 132-133.

The subclass mask bit is located in bit position 17 of control register 0. This bit is initialized to zero.

External Call

An interruption request for external call is generated when the CPU accepts the External Call order specified by a Signal Processor instruction addressing this CPU. The instruction may have been executed by this CPU or by another CPU. The request is preserved and remains pending in the receiving CPU until it is cleared. The pending request is cleared when it causes an interruption and by CPU reset.

Only one external call request, along with the processor address, may be held pending in a CPU at a time.

The condition is indicated by an external-interruption code of 1202 (hex). The processor address of the CPU that issued the Signal Processor instruction is stored at locations 132-133.

The subclass mask bit is located in bit position 18 of control register 0. This bit is initialized to zero.

Extended External Call

As interruption request for an extended external call is generated when the CPU accepts the Extended External Call order specified by a Signal Processor instruction addressing this CPU.

The request is preserved and remains pending in the receiving CPU until it is cleared.

The pending request is cleared when it causes an interruption and by CPU reset.

Facilities are provided for holding more than one extended external call pending, along with the processor address and parameter, in the receiving CPU.

The condition is indicated by an external interruption code of 1303. The processor address of the CPU that issued the Signal Processor instruction is stored at locations 132-133.

The sublcass mask bit is located in bit positon 28 of control register 0.

Interrupt Priority

When the CPU becomes enabled for more than one pending request the CPU applies the following procedure to determine which request or requests to indicate.

Each request is assigned a binary collating number, and the

request with the smallest collating number is taken. If several enabled requests have the same collating number, the interruption codes are ORed, and all are presented together. Those requests which are not enabled or which have a higher collating number are not presented and remain pending. The collating number is used only to determine which requests to present and is not preserved. The collating number consists of four fields, assigned as follows:

Name of Field	Collating number bits	Assigned From
Class number	0-3	Interruption Code bits 0-3
Subclass	4-11	All zeros if class number zero, Interruption code bits 8-15 if class number is not zero
Modifier	12-15	Interruption Code bits 4-7
Processor Address	16-31	Processor Address All zeros if interruption code bit is zero

APPENDIX C

Signal Processor Status Information

Seven status bits are defined whereby the processor addressed by a signal processor instruction can indicate its response to the designated function. The status bits and their bit positions in the general register are as follows:

bit position	status bit
8-23	unassigned, zeros stored
24	external call pending
25	stopped
26	operator intervening
27	check stop
28	not ready
29	unassigned, zero stored
30	invalid function
31	receiver check

Status bits 24-28 indicate the presence of the corresponding conditions in the addressed processor at the time the function code is received. The condition is indicated only in response to the sense function or when the condition precludes the successful execution of the designated function. Bit 31 indicates malfunctions detected during the execution of the instruction, and the signaling of the condition is not dependent on the function being initiated. The bits are defined as follows:

External Call Pending: This bit is set to one when an external call condition is pending for interruption in the addressed processor due to a previously issued Signal Processor instruction. The pending condition may be due to the same or another processor. The condition, when present, is indicated in response to sense of external call. Additionally, for external call it means that the requested interruption condition has not been generated.

Stopped: This bit is set to one when the addressed processor is in the stopped state and the function code specifies sense.

Operator Intervening: This bit is set to one when the addressed processor is executing certain operations initiated from the console or the remote operator control panel. The particular manually initiated operations that caused this bit to be turned on depends on the model and the function specified. The specified function cannot be performed and is not initiated. The operator- intervening status can be signaled in response to all functions.

Check Stop: This bit is set to one when the addressed processor is in the check stop state. The specified function cannot be performed and is not initiated. The condition, if

present, is indicated in response to all assigned functions except IMPL, program reset, and initial program reset.

Not Ready: This bit is set to one when the addressed processor uses reloadable control storage to perform the function and the required microprogram is not present. The function is not initiated. The condition, if present, is indicated in response to all assigned functions except initial microprogram load.

Invalid Function: This bit is set to one when the addressed processor receives an unassigned function code. No function is performed at the addressed processor. When the addressed processor is in the operator-intervening, check- stop, or not-ready state, either invalid function, the corresponding condition, or both are indicated.

Receiver Check: This bit is set to one when the addressed processor detects malfunctioning of equipment during the communications associated with the execution of signal processor, including receiving and decoding the function code. This condition can be signaled in response to any function code and indicates that the execution of the function has not been and will not be initiated. The other status bits are not necessarily valid. a machine-check condition may or may not have been generated at the addressed processor.

table: processor response to SIGP

	state of access path or processor								
	none of the other conditions	not ready	check-stop state	operator intervening	stopped	external call pending	processor temporarily busy	processor not operational	access path busy
function	v	v	v	v	v	v	v	v	v
sense	2	3	2	s	s	s	s	s	0
external call	2	3	2	s	0	s	s	s	0
emergency signal	2	3	2	0	0	s	s	s	0
start	2	3	2	0	0	s	s	s	0
stop	2	3	2	0	0	s	s	s	0
restart	2	3	2	0	0	s	s	s	0
initial program reset	2	3	0/2	0	0	0/s	0	s	0
program reset	2	3	0/2	0	0	0/s	0	s	0
stop and store status	2	3	2	0	0	s	s	s	0
initial microprogram load	2	3	0/2	0	0	0/s	0	0	0
unassigned function	2	3	2	i	i	i/s	i/s	i/s	i

0 - condition code 0 is set.
2 - condition code 2 is set.
3 - condition code 3 is set.
s - the corresponding status bit is indicated, and condition code 1 is set.
i - the invalid function status bit is indicated, and condition code 1 is set.
0/2, 0/s, i/s - either of the two indicated actions may be taken, depending on the situation and the model.

the order of the priority for the above actions is 2 due to access path busy, 3, 2 due to processor temporarily busy, s, i, 0.

APPENDIX D

Outline of Changes for IPC Implementation

In order to implement inter-process communication three CP modules were modified and a new module added. The user control block (VMBLOK) and the directory creation program were changed as well.

A brief description of the changes follows:

DMKPRV

Recognizes SIGP instruction and branches to DMKAES for signal processor simulation.

Simulates STAP - store processor address instruction.

DMKLOG

At logon time, assigns processor addresses to users which have 'DBSHARE' or 'MONITOR' privilege in their machine description.

DMKDSP

When reflecting external interrupts bits 6 and 7 of the interrupt code are tested. If bit 6 is one, the processor address is stored into location 132. If bit 7 is one, a one word parameter ('XINTPARM' from the XINTBLOK) is stored into location 128. Also the VMPEND bit in the VMBLOK is reset.

DMKDIR

Two new privilege classes have been defined as options:
DBSHARE x'04', and MONITOR x'01'.

These two new command privilege levels have been created to limit the use of the SIGP instruction, especially the Stop, Start, Reset and Restart orders.

VMBLOK

Two words in VMBLOK*, reserved for installation use have been redefined as follows:

```
VMPRS     DS  1F     receiver status
VMPADR    DS  1H     virtual processor address
VMPCLS    DS  1X     additional command level
*  bits defined in vmpcls
VMDBMON   EQU X'01'  special data base privilege
VMDBSHRE  EQU X'04'  standard data base privilege
```

PSA

External interrupt parameter defined.
INTEXP DS 1F SIGP parameter

| DMKHVA

Diagnose X'150' was added which given a userid in R0 and R1 returns the corresponding processor address in R1.

NEW MODULE -

DMKAES

This module simulates the Signal Processor instruction as defined in the Principles of Operation.

The VMBLOK of the receiving (virtual) processor is located. If the user is not on the system a condition code of 3 is returned. The issuing processor is checked for MONITOR or DBSHARE privilege. If neither is present, condition code 1 is returned. After the order code is checked for validity a branch is made to the appropriate routine.

On return from simulating the order, XINTQUE is called to set-up an external interrupt. A CPEXBLOK is then created and DMKSTKCP is called to stack the interrupt for deferred execution. Return is made via the Fast Reflect entry of the Dispatcher.

APPENDIX E

Outline of Changes for Segment Sharing Implementation

In order to implement Segment Sharing we added a new module (DMKSEG) and made some very minimal changes to two existing modules.

DMFCFC

Code has been added to recognize the SEGMENT command, as well as four additional QUERY command functions. The actual simulation of the commands is done in a new module - DMKSEG.

DMKVMA

The updates to this module prevent the release ('unshare') of a shared r/w segment, which is normally performed when it is detected that a shared page is changed or about to be changed (Store, Trace & Adstop commands).

NEW MODULE

DMKSEG

This module simulates the SEGment DEFine, RELease, LINK and DETach commands and the additional QUERY commands.

VM BLOCKS

The following block has been updated:

SHRTABLE - Named-Shared Segment Systems Table

An extension to this block adds two new fields:

```
***              SHRTABLE - EXTENSION FOR SHARED SEGMENTS
*
*             20   +----------------------+----------------------+
*                  |                    SHRPASSW                  |
*             28   +----------------------+----------------------+
*                  |                    SHROWNER                  |
*             30   +----------------------+----------------------+
*
***

SHRPASSW DS    CL8             SHARED SEGMENT PASSWORD
SHROWNER DS    CL8             SHARED SEGMENT OWNER'S USERID
SHRTBLSZ EQU   (*-SHRABLE)/8   SIZE OF SHRTABLE IN DOUBLEWORDS
```

DMKDIR

Two new privilege classes have been defined as options:
DBSHARE x'04', and MONITOR x'01'.

These two new command privilege levels have been created to limit the use of the SEGment commands, and the ability to apply SEGment commands to another machine.