PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 3 :		(11) International Publication Number: WO 86/ 00437
G06F 9/00, 9/46	A1	(43) International Publication Date: 16 January 1986 (16.01.86)

(21) International Application Number: PCT/US85/00735

(22) International Filing Date: 24 April 1985 (24.04.85)

(31) Priority Application Number: 626,363

(32) Priority Date: 28 June 1984 (28.06.84)

(33) Priority Country: US

(71) Applicant: MOTOROLA, INC. [US/US]; 1303 E. Algonquin Road, Schaumburg, IL 60196 (US).

(72) Inventors: ZOLNOWSKY, John ; 9 Homer Lane, Menlo Park, CA 94025 (US). CRUESS, Michael ; 7405 Cannon Mountain Place, Austin, TX 78749 (US). MacGREGOR, Douglas, B. ; 3705 Tarragona Lane, Austin, TX 78727 (US).

(74) Agents: GILLMAN, James, W. et al.; Motorola, Inc., Patent Department, Suite 300K, 4250 E. Camelback Road, Phoenix, AZ 85018 (US).

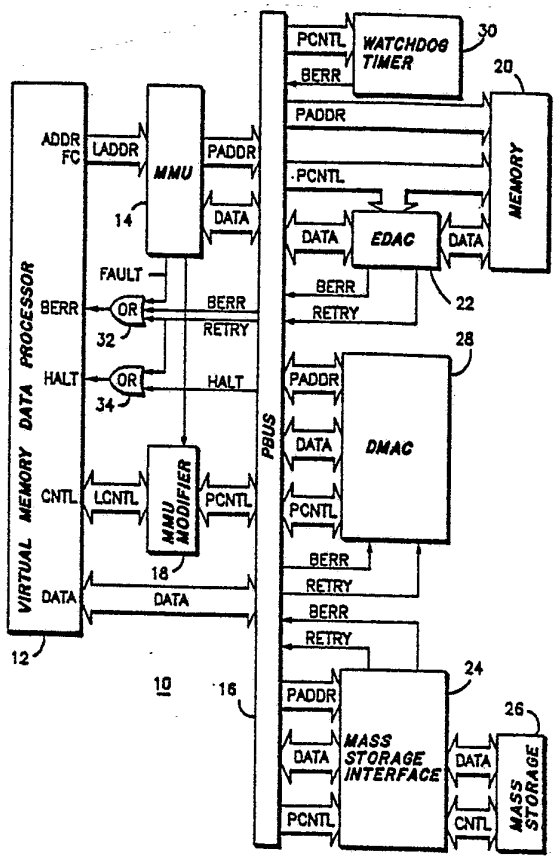
(81) Designated States: DE (European patent), FR (European patent), GB (European patent), IT (European patent), JP, KR, NL (European patent).

Published
With international search report.

(54) Title: DATA PROCESSOR HAVING MODULE ACCESS CONTROL

(57) Abstract

A data processor (12) cooperates with an access controller (14) to control access to a module stored in a storage device (20). In response to receiving an instruction which requests access to the module and specifies an address within the storage device (20) containing an access request, the data processor (12) retrieves the access request and provides the access request to the access controller (14). The data processor (12) will then initiate the requested access. However, the access will be faulted if the access controller (14) decides to deny the access request.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria
AU	Australia
BB	Barbados
BE	Belgium
BG	Bulgaria
BR	Brazil
CF	Central African Republic
CG	Congo
CH	Switzerland
CM	Cameroon
DE	Germany, Federal Republic of
DK	Denmark
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
HU	Hungary
IT	Italy
JP	Japan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
LI	Liechtenstein
LK	Sri Lanka
LU	Luxembourg
MC	Monaco
MG	Madagascar
ML	Mali
MR	Mauritania
MW	Malawi
NL	Netherlands
NO	Norway
RO	Romania
SD	Sudan
SE	Sweden
SN	Senegal
SU	Soviet Union
TD	Chad
TG	Togo
US	United States of America

DATA PROCESSOR HAVING MODULE ACCESS CONTROL

Field of the Invention

The present invention relates generally to a data processors and, more particularly, to a data processor having a module access control mechanism.

Background of the Invention

In many data processors, the executing program has the ability to access any address within the address space generally available to the processor. In many other data processors, access limitations are imposed upon a user program, but not upon the supervisor program. Typically, the access limitations are in the form of address range or space limits imposed by hardware. Another common limitation is the imposition of write protection upon certain designated address ranges which are otherwise accessible to the user program.

In some other systems, the supervisor program includes a number of service routines for performing input/output operations and other necessary system functions. In general, such routines are considered to be privileged, and all accesses thereto by user programs typically results in traps to an appropriate privilege violation handler within the supervisor program. In such implementations, the handler is responsible for deciding if the request should be honored. If the decision is affirmative, the handler enables the requested service to be performed before control is transferred back to the user program. While this software implemented access control mechanism is quite versatile, the overhead associated with such a mechanism is far from insignificant.

In some other data processors, such as the Digital Equipment Corporation VAX and the National Semiconductor NSC16000 microprocessor, a program may be configured as a set of data/code modules which can be "called" as appropriate by

other modules. At the end of the called module, control is returned to the calling module. However, these processors provide no mechanism for controlling access to such modules. Thus, the module call instruction is comparable to a conventional branch-to-subroutine instruction wherein the data processor would simply stack away onto a user stack certain return information before branching to the appropriate starting address of the module. In some of these systems, this starting address is part of a "module descriptor" which is constructed by the compiler/assembler and linker in the process of creating an executable load module. Other information relating to the module may also be provided in the module descriptor.

In the General Electric GE645 "MULTICS" machine, and, more recently, in some of the machines offered commercially by Prime Computers and Data General, each "page" of the available address space within the system memory has associated with it an access level, creating in effect a set of concentric "rings" of protection. Although the number of rings may vary, the most sensitive data/code modules are typically stored within the innermost ring and the user modules are within the outermost ring, the balance of the supervisor program and associated compilers/assemblers being appropriately distributed among the several available rings. In order to obtain access to data/code modules stored within the innermost ring, the calling module must have been granted the highest access level, while even those modules having the lowest access level can access modules stored in the outermost ring. In this more useful form, the call module instruction allows a user program controlled access to data/code modules which the system wishes to protect against unauthorized use.

In a typical data processing system which implements an access control mechanism, the supervisor program has the responsibility of assigning access levels to each of the user programs which are installed in the system. For example, some

users, because of their duties, may be assigned a higher access privilege than other users of the same program. Similarly, different programs, because of their nature, may require higher access levels than other programs. On the other hand, all of the programs will typically require access to those modules of the supervisor program which perform common input/output and related service functions. The call module mechanism facilitates just such a dynamic change in access level.

During the compilation/assembly and linking process, the supervisor program typically initializes the module descriptors (sometimes referred to as segment descriptors) to contain information relating to the address of the respective modules and to the access level thereof. Depending upon the requirements of the system, these descriptors may be stored either within or without the ring containing the respective modules. The addresses of these descriptors are thereafter inserted into the appropriate call module instructions in the calling module and the linked program installed into an appropriate storage medium within the system resources. Thus, whenever the program is executed, the supervisor program can be sure that all module calls made by that program have previously been approved. However, the program must still be prevented from extending the higher level access privilege beyond the authorized module. This dynamic access control function is typically handled by an access controller implemented within the data processor itself or in a memory management unit which is tightly coupled to the data processor. In general, the access controller monitors each access to the system storage to determine that the access level of the currently executing module is greater than or equal to the access level of the accessed page. If so, the access is allowed; if not, the access is faulted to force the termination of the calling module. Whenever a call module instruction is executed, the data processor notifies the

access controller that the access level must be changed to a higher level, if necessary, to enable the called module to execute. The access controller would thereafter allow accesses to pages having the higher access level. Upon executing a corresponding "return-from-module" instruction, the data processor orders the access control to change the access level back to the original level of the calling module.

In some systems, each access level has a set of "gates" associated therewith, each of which can be "open" or "closed" at the discretion of the supervisor program. In general, if a particular module is going to be made accessible to user programs having a particular access level, the supervisor program will open a gate to that module by storing the descriptor for that module within a particular gate table at that access level; without such an entry, the gate will be effectively closed. Thereafter, a calling module can request access to a module by specifying the number of the gate within the calling module's access level which controls access to the desired module, together with the index into the respective gate table at which the module descriptor is stored. If the access controller verifies that such an entry actually exists, the processor is allowed to establish the appropriate access level and pathway to the called module using the information contained in the module descriptor identified in the call module instruction; otherwise the access is faulted to force the termination of the calling module. Upon exiting from the called module, the processor reestablishes the original access level of the calling module before returning control thereto. In addition to dedicating significant storage space for the gate tables, this technique requires a significant amount of rather complex circuitry to implement the table lookup function.

Summary of the Invention

Accordingly, it is an object of the present invention to

provide a data processor having an improved module access control mechanism.

Another object is to provide a module access control mechanism which does not require the data processor to be concerned with the criteria under which access is granted.

Yet another object is to provide a data processor wherein an access controller independent of the data processor decides for the data processor whether a requested access should be granted.

Still another object is to provide an efficient mechanism for a data processor to cooperate with an independent access controller in the control of access to a module stored in system storage.

One other object of the present invention is to provide an improved gate mechanism for an access controller to directly control access to the system storage by modules executing in a data processor.

These and other objects are achieved in a data processor which has been adapted in accordance with the present invention to cooperate with an access controller to control access to a module stored in a storage device. In the most basic form of the present invention, the data processor is constructed to receive an instruction which requests accesss to the module, the instruction specifying an address within the storage device containing an access request. Using the address specified in the instruction, the data processor retrieves the access request and provides the access request to the access controller. The data processor then initiates the requested access to the module. However, the access will be faulted if the access controller decides to deny the access request.

In the preferred form of the present invention, the data processor requests the decision of the access controller to the access request before attempting the requested access. If the decision of the access controller is affirmative, the data

processor allows access to the module. However, if the decision of the access controller is negative, the data processor denies access to the module.

In either form, the data processor need not be aware of the access criteria being imposed by the access controller. Thus, the form and content of the access request may be changed to suit specific requirements without changing the data processor and the manner in which the access control mechanism is implemented therein.

Brief Description of the Drawings

Figure 1 is a block diagram of a data processing system suitable for practicing the present invention.

Figure 2 is a block diagram of the data processor of Figure 1.

Description of the Invention

Shown in Figure 1 is a data processing system 10 wherein logical addresses (LADDR) issued by a data processor (DP) 12 are mapped by a memory management unit (MMU) 14 to a corresponding physical address (PADDR) for output on a physical bus (PBUS) 16. Simultaneously, the various logical access control signals (LCNTL) provided by DP 12 to control the access are converted to appropriately timed physical access control signals (PCNTL) by a modifier unit 18 under the control of MMU 14. In the preferred form, DP 12 is adapted in accordance with the present invention to cooperate with an access controller implemented, for example, in MMU 14, to control access to data and code stored as modules in the memory 20.

In response to a particular range of physical addresses (PADDR), memory 20 will cooperate with an error detection and correction circuit (EDAC) 22 to exchange data (DATA) with DP 12 in synchronization with the physical access control signals (PCNTL) on PBUS 16. Upon detecting an error in the data, EDAC

22 will either signal a bus error (BERR) or request DP 12 to retry (RETRY) the exchange, depending upon the type of error.

In response to a different physical address, mass storage interface 24 will cooperate with MP 12 to transfer data to or from mass storage 26. If an error occurs during the transfer, interface 24 may signal a bus error (BERR) or, if appropriate, request a retry (RETRY).

In the event that the MMU 14 is unable to map a particular logic address (LADDR) into a corresponding physical address (PADDR), the MMU 14 will signal an access fault (FAULT). As a check for MMU 14, a watchdog timer 28 may be provided to signal a bus error (BERR) if no physical device has responded to a physical address (PADDR) within a suitable time period relative to the physical access control signals (PCNTL).

If, during a data access bus cycle, a RETRY is requested, OR gates 30 and 32 will respectively activate the BERR and HALT inputs of DP 12. In response to the simultaneous activation of both the BERR and HALT inputs thereof during a DP-controlled bus cycle, DP 12 will abort the current bus cycle and, upon the termination of the RETRY signal, retry the cycle.

If desired, operation of DP 12 may be externally controlled by judicious use of a HALT signal. In response to the activation of only the HALT input thereof via OR gate 32, DP 12 will halt at the end of the current bus cycle, and will resume operation only upon the termination of the HALT signal.

In response to the activation of only the BERR input thereof during a processor-controlled bus cycle, DP 12 will abort the current bus cycle, internally save the contents of the status register, enter the supervisor state, turn off the trace state if on, and generate a bus error vector number. DP 12 will then stack into a supervisor stack area in memory 20 a block of information which reflects the current internal context of the processor, and then use the vector number to branch to an error handling portion of the supervisor program.

During the stacking operation, DP 12 will stack certain information of a general nature, including: the saved status register, the current contents of the program counter, the contents of the instruction register which is usually the first word of the currently executing instruction, the logical address which was being accessed by the aborted bus cycle, and the characteristics of the aborted bus cycle, i.e. read/write, instruction/data and function code. In addition to the above information, DP 12 is constructed to stack much more information about the internal machine state. If the exception handler is successful in resolving the error, the last instruction thereof will return control of DP 12 to the aborted program. During the execution of this instruction, the additional stacked information is retrieved and loaded into the appropriate portions of DP 12 to restore the state which existed at the time the bus error occurred.

The preferred operation of DP 12 will be described with reference to Figure 2 which illustrates the internal organization of a microprogrammable embodiment of DP 12. Since the illustrated form of DP 12 is very similar to the Motorola MC68000 microprocessor described in detail in the several U.S. Patents cited hereafter, the common operation aspects will be described rather broadly. Once a general understanding of the internal architecture of DP 12 is established, the discussion will focus on the access control aspect of the present invention.

The DP 12 is a pipelined, microprogrammed data processor. In a pipelined processor, each instruction is typically fetched during the execution of the preceding instruction, and the interpretation of the fetched instruction usually begins before the end of the preceding instruction. In a microprogrammed data processor, each instruction is typically fetched during the execution of the preceding instruction, and the interpretation of the fetched instruction usually begins before the end of the preceding instruction. In a

microprogrammed data processor, each instruction is executed as a sequence of microinstructions which perform small pieces of the operation defined by the instruction. If desired, user instructions may be thought of as macroinstructions to avoid confusion with the microinstructions. In the DP 12, each microinstruction comprises a microword which controls microinstruction sequencing and function code generation, and a corresponding nanoword which controls the actual routing of information between functional units and the actuation of special function units within DP 12. With this in mind, a typical instruction execution cycle will be described.

At an appropriate time during the execution of each instruction, a prefetch microinstruction will be executed. The microword portion thereof will, upon being loaded from micro ROM 34 into micro ROM output latch 36, enable function code buffers 38 to output a function code (FC) portion of the logical address (LADDR) indicating an instruction cycle. Upon being simultaneously loaded from nano ROM 40 into nano ROM output latch 42, the corresponding nanoword requests bus controller 44 to perform an instruction fetch bus cycle, and instructs execution unit 46 to provide the logical address of the first word of the next instruction to address buffers 48. Upon obtaining control of the PBUS 16, bus controller 44 will enable address buffers 48 to output the address portion of the logical address (LADDR). Shortly thereafter, bus controller 44 will provide appropriate data strobes (some of the LCNTL signals) to activate memory 20. When the memory 20 has provided the requested information, bus controller 44 enables instruction register capture (IRC) 50 to input the first word of the next instruction from PBUS 16. At a later point in the execution of the current instruction, another microinstruction will be executed to transfer the first word of the next instruction from IRC 50 into instruction register (IR) 52, and to load the next word from memory 20 into IRC 50. Depending upon the type of instruction in IR 52, the word in IRC 50 may

be immediate data, the address of an operand, or the first word of a subsequent instruction. An example of an instruction set which is generally suitable for DP 12, and the microinstruction sequences which may be adapted to implement such an instruction set, are set forth fully in U.S. Patent No. 4,325,121 entitled "Two Level Control Store for Microprogrammed Data Processor" issued 13 April 1982 to Gunter et al, and which is hereby incorporated by reference.

As soon as the first word of the next instruction has been loaded into IR 52, address 1 decoder 54 begins decoding certain control fields in the instruction to determine the micro address of the first microinstruction in the initial microsequence of the particular instruction in IR 52. Simultaneously, illegal instruction decoder 56 will begin examining the format of the instruction in IR 52. If the format is determined to be incorrect, illegal instruction decoder 56 will provide the micro address of the first microinstruction of an illegal instruction microsequence. In response to the format error, exception logic 58 will force multiplexor 60 to substitute the micro address provided by illegal instruction decoder 56 for the micro address provide by address 1 decoder 54. Thus, upon execution of the last microinstruction of the currently executing instruction, the microword portion thereof may enable multiplexor 60 to provide to an appropriate micro address to micro address latch 62, while the nanoword portion thereof enables instruction register decoder (IRD) 64 to load the first word of the next instruction from IR 52. Upon the selected micro address being loaded into micro address latch 62, micro ROM 34 will output a respective microword to micro ROM output latch 36 and nano ROM 40 will output a corresponding nanoword to nano ROM output latch 42.

Generally, a portion of each microword which is loaded into micro ROM output latch 36 specifies the micro address of the next microinstruction to be executed, while another

portion determines which of the alternative micro addresses will be selected by multiplexor 60 for input to micro address latch 62. In certain instructions, more than one microsequence must be executed to accomplish the specified operation. These tasks, such as indirect address resolution, are generally specified using additional control fields within the instruction. The micro addresses of the first microinstructions for these additional microsequences are developed by address 2/3 decoder 66 using control information in IR 52. In the simpler form of such instructions, the first microsequence will typically perform some preparatory task and then enable multiplexor 60 to select the micro address of the microsequence which will perform the actual operation as developed by the address 3 portion of address 2/3 decoder 66. In more complex forms of such instructions, the first microsequence will perform the first preparatory task and then will enable multiplexor 60 to select the micro address of the next preparatory microsequence as developed by the address 2 portion of address 2/3 decoder 66. Upon performing this additional preparatory task, the second microsequence then enables multiplexor 60 to select the micro address of the microsequence which will perform the actual operation as developed by the address 3 portion of address 2/3 decoder 66. In any event, the last microinstruction in the last microsequence of each instruction will enable multiplexor 60 to select the micro address of the first microinstruction of the next instruction as developed by address 1 decoder 54. In this manner, execution of each instruction will process through an appropriate sequence of microinstructions. A more thorough explanation of a suitable micro address sequence selection mechanism is given in U.S. Patent No. 4,342, 078 entitled "Instruction Register Sequence Decoder for Microprogrammed Data Processor" issued 27 July 1982 to Tredennick et al, and which is hereby incorporated by reference.

In contrast to the microwords, the nanowords which are loaded into nano ROM output latch 42 indirectly control the routing of operands into and, if necessary, between the several registers in the execution unit 46 by exercising control over register control (high) 68 and register control (low and data) 70. In certain circumstances, the nanoword enables field translation unit 72 to extract particular bit fields from the instruction in IRD 64 for input to the execution unit 46. The nanowords also indirectly control effective address calculations and actual operand calculations within the execution unit 46 by exercising control over AU control 74 and ALU control 76. In appropriate circumstances, the nanowords enable ALU control 76 to store into status register (SR) 78 the condition codes which result from each operand calculation by execution unit 46. A more detailed explanation of a suitable form of ALU control 76 is given in U.S. Patent No. 4,312,034 entitled "ALU and Condition Code Control Unit for Data Processor" issued 19 January 1982 to Gunter, et al, and which is hereby incorporated by reference. Other details relating to the construction and operation of DP 12 may be found in US Application Serial Number 447,600 entitled "Data Processor Version Validation" filed 7 December 1982 and allowed on 21 June 1984.

Since DP 12 is a microprogrammed machine, the implementation of additional instructions is primarily a matter of providing appropriate microsequences for the new instructions, provided, of course, that all of the resources and control paths are available to support the functionality of the new instructions. Such is the case of the module call (CALLM) and module return (RTM) instructions in accordance with the present invention, since the only hardware requirement imposed upon DP 12 by this instruction is the existing ability to read from and write to specific addresses within the overall address space already available to DP 12. On the other hand, within the constraints imposed by the

CALLM/RTM interface, the implementation of the access controller function is totally at the discretion of the system designer. Thus, for the purposes of describing the operation of DP 12 in the execution of the CALLM and RTM instructions, the access controller, which could be conveniently integrated into the MMU 14, for example, will be assumed to exist as a "black box" which DP 12 perceives as a set of several registers accessible at respective predetermined addresses within the existing address space.

In the preferred form, the CALLM instruction consists of an effective address which specifies the address within the memory 20 at which a descriptor for the called module may be found, and an argument count which indicates the number of arguments, if any, the calling module is passing to the called module. In preparation for the CALLM instruction, the module descriptor will have been initialized at link time by the supervisor program to contain the entry address of the called module and the address of the data area associated with that module. The module descriptor may also contain the address of a stack upon which the module expects to find the arguments. In addition, the module descriptor will contain an access request of a specific format appropriate for the particular level of access control desired by the designer of the system. For example, in the preferred embodiment, the access request consists of an access type code which indicates whether the access level must be changed, and, if so, what new access level the called module requires.

Upon receiving the CALLM instruction for execution, DP 12 will first evaluate the effective address and then retrieve from that address the access request, the module address and the module data area address. DP 12 then tests the access request to determine the type of access which is to be made, that is, whether an access level change is required or the current access level is adequate for the called module. In addition, the preferred form of the access request also

indicates whether the called module expects to find the arguments on the calling module's stack or on the called module's stack.

If, for example, the access type indicates that the access level need not be changed, DP 12 will build a module stack frame at the top of the current stack. If the called module expects to find the arguments on the calling module's stack, DP 12 will stack the calling module's stack pointer on the module stack frame so that the called module will know where to find the arguments. If the called module expects to find the arguments on its own stack, DP 12 does not stack the calling module's stack pointer, but simply advances the module stack frame pointer to compensate for the shortcut. DP 12 then writes the current value of the calling module's program counter on the module stack frame, followed by the address of the module descriptor.

In the preferred form, the first word of the called module specifies a particular one of the several registers within DP 12 which that module expects to contain the address of the data area of that module. At this point in the execution of the CALLM instruction, DP 12 will retrieve this register specifier, and then store the current contents of the specified register on the module stack frame. DP 12 completes the module stack frame by storing the argument count specified by the CALLM instruction and the access request retrieved from the module descriptor. DP 12 then begins execution of the module at the first instruction following the register specifier.

If, on the other hand, the access type indicates that the access level must be changed, DP 12 will first determine if the calling module is passing arguments to the called module and, if so, DP 12 will verify that all of the arguments are within the legitimate address space of the calling module. If an access violation is detected, DP 12 will force the termination of the calling module by vectoring to an exception

handler. If no access violation is detected, DP 12 will read what it believes to be the access level of the calling module from a "current access level register" known to the DP 12 only as a first predetermined address within the address space. DP 12 will then write the address of the called module to a "module address register" known to the DP 12 only as a second predetermined address in the available address space, and the "new" access level to a "increase access level register" known to the DP 12 only as a third predetermined address within the address space. DP 12 then reads what it believes to be the decision of the access controller to the access request from an "access status register" known to the DP 12 only as a fourth predetermined address within the address space.

If the decision is negative (at least what DP 12 perceives to be negative), DP 12 will force the termination of the calling module by vectoring to the exception handler. On the other hand, if the decision is perceived by DP 12 to be affirmative, DP 12 will insert the "old" access level into the access request being maintained within a temporary register within DP 12 in place of the "new" access level originally contained therein.

If the called module expects to find the arguments on the calling module's stack or at least a pointer to the arguments within the module stack frame, DP 12 procedes to complete the module stack frame just as in the case described above when there was no access level change. On the other hand, if the called module expects to find the arguments on its own stack, DP 12 will retrieve the called module's stack pointer from the module descriptor, and transfer all of the arguments from the calling module's stack to the called module' stack. DP 12 then builds the module stack frame as described above but on the called module's stack rather than on the calling module's stack. In either case, after the module stack frame is complete, DP 12 then begins execution of the module at the first instruction following the register specifier.

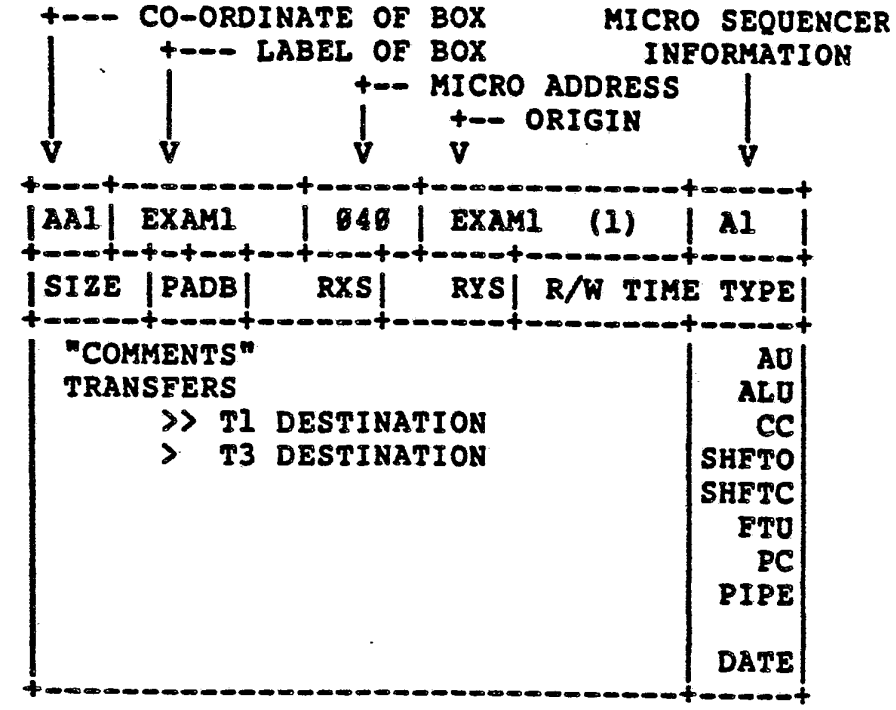
Upon receiving the RTM instruction for execution at the end of the called module, DP 12 will retrieve the access request, the argument count, the program counter for the calling module and the value which was in the register used by the called module as the pointer to its data area. If the access type in the access request indicates that no access change was made, DP 12 adjusts the current stack pointer to discard the module stack frame and any associated arguments, restores the original value of the register used by the called module, and then restores the program counter to resume execution of the calling module. If, however, the access type indicates that an access level change was made, DP 12 retrieves the "old" stack pointer from the called module's stack, before writing the "old" access level to a "decrease access level register" known to DP 12 only as a fifth predetermined address within the address space. DP 12 then reads the "access status register" again to see what the decision of the access controller is to the access level decrease request. If the decision is negative, DP 12 will force the termination of the calling module by vectoring to the exception handler. If the decision is affirmative, DP 12 will adjust the "old" stack pointer to discard the module stack frame and the associated arguments to derive the proper current stack pointer. DP 12 will then procede as described above to restore the original value of the register used by the called module, and then the program counter to resume execution of the calling module.

As explained above, DP 12, in the course of processing the CALLM and RTM instructions, waits for the decision of the access controller before proceeding with the execution of the called module. However, if desired, DP 12 could simply proceed with the requested access after passing the access request to the access controller. If the access controller decides to deny access, the access controller can simply fault the access cycle, thereby forcing DP 12 into the exception

handler anyway. Thus, the present invention, in a general sense, relates to a mechanism for a data processor such as DP 12 to advise an independent access controller that an access request is going to be made unless the access controller prevents it. How the access controller decides whether or not to allow the access is totally outside the scope of the data processor.

Using the guide shown in Appendix I, the detailed microsequence shown in Appendix II for a preferred implementation of the CALLM and RTM instructions in a modified form of DP 12 may be understood. For a general understanding of such microsequences, as well the microsequences for all of the instructions in DP 12, reference may be made to US Patent Number 4,325,121.

MICROINSTRUCTION LISTING



ORIGIN: if shared, co-ordinate of origin
if origin, # of boxes sharing with this box

DATA ACCESS INFORMATION:

R/W
- . - no access
- <W> - write
- <> - read
- SPC - special signal
- EXL - latch exception

TIME
- X - no timing associated
- T1 - write to aob in T1
- T3 - write to aob in T3
- T0 - aob writen before T1

TYPE

.,<>,<W> on R/W
- . - normal access
- UNK - program/data access
- CNORM - conditional normal
- CUNK - conditional prog/data
- AS - alternate address space
- CPU1 - cpu access - different bus error
- CPU2 - cpu access - normal bus error
- RMC - read-modify-write access

SPC on R/W
- RST1 - restore stage 1
- RST2 - restore stage 2
- HALT - halt pin active
- RSET - reset pin active
- SYNC - synchronize machine

EXL on R/W
- BERR - bus error
- AERR - address error
- PRIV - privilege viol.
- TRAC - trace

LINA	- line a	TRAP	- trap
LINF	- line f	COP	- protocol viol.
ILL	- illegal	FORE	- fomat error
DVBZ	- divide by zero	INT	- interrupt 1st stack
BDCK	- bad check	INT2	- interrupt 2nd stack
TRPV	- trap on overflow	NOEX	- no exception

MICRO SEQUENCER INFORMATION:

DB - direct branch - next microaddress in microword
BC - conditional branch
A1 - use the A1 PLA sample interrupts and trace
A1A - use the A1 PLA sample interrupts, do not sample trace
A1B - use the A1 PLA do not sample interrupts or trace
A2 - use the A2 PLA
A7 - functional conditional branch (DB or A2 PLA)
A4 - use the A4 latch as next micro address
A5 - use the A5 PLA
A6 - use the A6 PLA

SIZE:

size =	byte	nano specified constant value
size =	word	nano specified constant value
size =	long	nano specified constant value
size =	ircsz	irc[11]=0/1 => word/long
size =	irsz	ird decode of the instruction size (byte/word/long). Need to have file specifying residual control
size =	ssize	shifter control generates a size value. The latch in which this value is held has the following encoding 000 = byte 001 = word 010 = 3-byte 011 = long 100 = 5-byte *** must act as long sized

RXS - RX SUBSTITUTIONS:

RX is a general register pointer. It is used to point at either special purpose registers or user registers. RX generally is used to translate a register pointer field within an instruction into the control required to select the the appropriate register.

rx = rz2d/rxd	conditionally substitute rz2d use rz2d and force rx[3]=0	
	mul.l	0100 110 000 xxx xxx
	div.l	0100 110 001 xxx xxx

```
rx = rx          ird[11:9] muxed onto rx[2:0]
                 rx[3] = 0 (data reg.)
                     (unless residual points)
                 rxa then rx[3] = 1
                     (residual defined in opmap)

rx = rz2         irc2[15:12] muxed onto rx[3:0]
                 rx[3] is forced to 0 by residual control
                   div.l            0100 110 001 xxx xxx
                   bit field reg    1110 1xx 111 xxx xxx

rx = rp          rx[3:0] = ar[3:0]
                 The value in the ar latch must be
                 inverted before going onto the rx bus
                 for movem rl,-(ry) 0100 100 01x 100 xxx

rx = rz          irc[15:12] muxed onto rx[3:0]
                 (cannot use residual control)

rx = ro2         rx[2:0] = irc2[8:6]
                 rx[3] = 0 (data reg.)
                 Used in Bit Field, always data reg

rx = car         points @ cache address register
rx = vbr         points @ vector base register

rx = vatl        points @ vatl

rx = dt          points @ dt

rx = crp         rx[3:0] = ar[3:0]
                 The value in ar points at a control
                 register (i.e. not an element of the
                 user visible register array)

rx = usp         rx[3:0] = F
                 force effect of psws to be negated (0)

rx = sp          rx[2:0] = F,
                    if psws=0 then address  usp
                    if psws=1 & pswm=0 then isp
                    if psws=1 & pswm=1 then msp
```

RYS - RY SUBSTITUTIONS:

```
ry = ry          ird[2:0] muxed onto ry[2:0]
                 ry[3] = 1 (addr reg.) unless residual
                    points
                 ryd then ry[3] = 0. (residual defined
                    in opmap)

ry = ry/dbin     This is a conditional substitution
     ry/dob      for the normal ry selection (which
```

includes the residual substitutions like dt) with dbin or dob. The substitution is made based on residual control defined in opmap (about 2 ird lines) which selects the dbin/dob and inhibits all action to ry (or the residually defined ry). Depending upon the direction to/from the rails dbin or dob is selected. If the transfer is to the rails then dbin is substituted while if the transfer is from the rails dob is substituted.

Special case: IRD = 0100 0xx 0ss 000 xxx (clr,neg,negx,not) where if driven onto the a-bus will also drive onto the d-bus.

ry = rw2 irc2[3:0] muxed onto ry[3:0]

use rw2

movem ea,rl	0100 110 01x xxx xxx
div.l	0100 110 001 xxx xxx
bfield	1110 xxx xxx xxx xxx
cop	1111 xxx xxx xxx xxx

do not allow register to be written to

div.w	1000 xxx x11 xxx xxx

force ry[3] = 0

div.l	0100 110 001 xxx xxx
bfield	1110 1xx x11 xxx xxx

ry = rw2/dt conditionally substitute rw2 or dt

use rw2 and force ry[3]=0

mul.l	0100 110 000 xxx xxx
	and irc2[10] = 1
div.l	0100 110 001 xxx xxx
	and irc2[10] = 1

ry = vdtl points @ virtual data temporary

ry = vat2 points @ virtual address temporary 2

ry = dty points @ dt

AU - ARITHMETIC UNIT OPERATIONS:

0- ASDEC add/sub add/sub based on residual control
sub if ird = xxxx xxx xxx 100 xxx

1- ASXS add/sub add/sub based on residual (use alu add/sub). Do not extend db entry

add if ird = 1101 xxx xxx xxx xxx add

or 0101 xxx 0xx xxx xxx addq

2- SUB	sub	subtract AB from DB
3- DIV	add/sub	do add if aut[31] = 1, sub if aut[31] = 0; take db (part rem) shift by 1 shifting in alut[31] then do the add/sub.
4- NIL		
6- SUBZX	sub	zero extend DB according to size then sub AB
8- ADDX8	add	sign extend DB 8 -> 32 bits then add to AB
9- ADDX6	add	sign extend DB 16 -> 32 bits then add to AB
10- ADD	add	add AB to DB
11- MULT	add	shift DB by 2 then add constant sign/zero extend based on residual and previous aluop muls = always sxtd mulu = sxtd when sub in previous aluop
12- ADDXS	add	sign extend DB based on size then add to AB
13- ADDSE	add	sign extend DB based on size then shift the extended result by 0,1,2,3 bits depending upon irc[10:9]. Finally add this to AB
14- ADDZX	add	zero extend DB according to size then add to AB
15- ADDSZ	add	zero extend DB according to size, shift by 2, then add

CONSTANTS

0,1	1 selected by: (div * allzero) + (mult * alu carry = 0)
1,2,3,4	selected by size byte = 1 word = 2 3-by = 3 long = 4

If (Rx=SP or Ry=SP) and (Ry=Ry or Rx=Rx) and (Rx or Ry is a source and destination) and (au constant = 1,2,3,4) and (size = byte) then constant = 2 rather than one.

ALU - ARITHMETIC AND LOGIC UNIT OPERATIONS:

col0 = x,nil
col1 = and
col2 = alu1,div,mult,or
col3 = alu2,sub

row		col 1	col 2	col 3
1	ADDROW	and	add	
2	ADDXROW	and	addx	add
3	SUBROW	and	sub	
4	SUBXROW	and	subx	add1
5	DIVROW	and	div	sub
6	MULTROW	and	mult	sub
7	ANDROW	and	and	
8	EORROW	and	eor	
9	ORROW	and	or	add
10	NOTROW	and	not	
11	CHGROW	and	chg	
12	CLRROW	and	clr	
13	SETROW	and	set	

		cin
add	db + ab	0
addx	db + ab	x
add1	db + ab	1
and	ab ^ db	-
chg	ab xor k=-1	-
clr	ab ^ k=0	-
eor	ab xor db	-
not	~ab v db	-
or	ab v db	-
set	ab v k=-1	-
sub	db + ~ab	1
subx	db + ~ab	x

mult (db shifted by 2) add/sub (ab shifted by 0,1,2 (if 0 then add/sub 0)) control for add/sub and shift amount comes from regb. Don't assert atrue for mult

cin = 0

div build part. quot and advance part. remain.1 ab (pr.1:pq) shifted by 1, add0, value shifted in = au carry (quot bit)

cin = 0

must assert atrue for div

The condition codes are updated during late T3 based upon the data in alut and/or rega. These registers can be written to during T3. In the case of rega, there are times when the value to be tested is the result of an insertion from regb.

CC - CONDITION CODE UPDATE CONTROL:

row		col 1		col 2		col 3	
1	add	cnzvc		ddddd		ddddd	
2	addx	cnzvc		ddkdc	(bcd1)	cdzdc	(bcd2)
3	sub	cnzvc		knzvc	(cmp)	ddddd	
4	subx	cnzvc		ddkdc	(bcd1)	cdzdc	(bcd2)
5	div	knzv0	(div)	ddddd		ddddd	
6	mull	knzv0		ddddd		ddddd	
7	rotat	knz0c		ddddd		ddddd	
8	rox	cnz0c		knz00		kkkvk	
9	bit,bitfld	kkzkk	(bit)	knz00	(bfld1)	kkzkk	(bfld2)
10	log	knz00		ddddd		ddddd	

```
standard
  n = alut msb (by size)
  z = alut=0 (by size)

non-standard
  add      c = cout
           v = vout
  addx.1   c = cout
           z = pswz ^ locz
           v = vout
  bcd1     c = cout
  bcd2     c = cout v pswc
           z = pswz ^ locz
  bfld1    n = shiftend
           z = all zero
  bfld2    z = pswz ^ allzero
  bit      z = allzero
  div      v = au carry out
  mull     n = (shiftend ^ irc2[10]) v
                (alut[31] ^ ~irc2[10])
           z = (alut=0 ^ shift allzero ^ irc2[10]) v
                (alut=0 ^ ~irc2[10])
           v = ~irc2[10] ^ ((irc2[11] ^ (~allzero ^
                ~alut[31]) v (~allone ^ alut[31])) v
                 (~irc2[11] ^ ~allzero))
  rotat    c = shiftend = (sc=0 - 0  sc<>0 - end)
  rox.1    c = shiftend = (sc=0 - pswx  sc<>0 - end)
               ! can do this in two steps as knz0c where
               ! c=pswx and cnz0c where c=shiftend (not
               ! with share row with shift)
  rox.3    v = shift overflow = ((~allzero ^ sc>sz) v
                (~(allzero v allones) ^ sc<=sz))
```

```
                ! can simplify this if we don't share
                ! rows but it will cost another box
sub.1     c = ~cout
          v =  vout
sub.2     c = ~cout
          v =  vout
subx.1    c = ~cout
          z = pswz ^ locz
          v =  vout
subx.2    c = ~cout
subx.3    c = ~cout v pswc
          z = pswz ^ locz
```

The meaning and source of signals which are used to set the condition codes is listed below:

allzero = every bit in rega field = 0 where the field is defined as starting at the bit pointed to by start and ending (including) at the bit pointed to by end. (see shift control)

allone = every bit in rega field = 1 where the field is defined as starting at the bit pointed to by start and ending (including) at the bit pointed to by end. (see shift control)

shiftend = the bit in rega pointed to by end = 1. (see shift control)

locz = all alut for the applicable size = 0.

SHFTO - SHIFTER OPERATIONS:

ror value in rega is rotated right by value in shift count register into regb.

sxtd value in rega defined by start and end registers is sign extended to fill the undefined bits and that value is rotated right by the value in the shift count register. The result is in regb.

xxtd value in rega defined by start and end registers is PSWX extended to fill the undefined bits and that value is rotated right by the value in the shift count register. The result is in regb.

zxtd value in rega defined by start and end registers is zero extended to fill the undefined bits and that value is rotated right by the value in the shift count register. The result is in regb.

ins the value in regb is rotated left by the value in shift count register and then inserted into the field defined by the start and end register in rega. Bits in rega that are not defined by start and end are not modified.

boffs provides the byte offset in regb. If irc2[11]=1 then the offset is contained in R0 and as such rega should be sign extended from rega to regb using the values established in start, end, and shift count of 3,31,3 respectively. If irc2[11]=0 then the offset is contained in the immediate field and should be loaded from irc2[10:6] or probably more conveniently osr[4:0]. This value however should be shifted by 3 bits such that osr[4:3] are loaded onto regb[1:0] with zero zero extension of the remaining bits.

offs provides the offset in regb. If irc2[11]=1 then the offset is contained in R0 and as such DB>REGB should be allowed to take place. If irc2[11]=0 then the offset is contained in the immediate field and osr[4:0] should be loaded onto regb[4:0] with zero extension of the remaining bits.

SHFTC - SHIFTER CONTROL:

	{sbm1}		{sbm2}
BIT	st = 0		st = wr - 8
bit	en = -1	(31)	en = wr - 1
mvp	sc = wr	(16,32)	sc = wr - 8
swap	wr = BC[12:7]	(16,32)	wr = wr - 8
callm	osr = x		osr = x
	cnt = x		cnt = x

{sbm3}	{sbm4}
st = DB [5:0] mod sz	st = 0
en = DB [5:0] mod sz	en = -1 (31)
sc = 0	sc = wr
wr = DB [5:0]	wr = wr
osr = x	osr = x
cnt = x	cnt = x

{sbm5}	{sbm6}
st = x	st = 16
en = x	en = 31
sc = x	sc = 16
wr = DB [7:2]	wr = wr - 1
osr = x	osr = x
cnt[1:0] = DB [1:0]	cnt = x

{}
st = x
en = x
sc = x

```
        wr = x
        osr = x
        cnt = x

        {mul1}                                  {mul2}
MUL     st = wr                                 st = wr - 2
mulw    en = -1 mod sz      (15,31)             en = wr
mull    sc = wr                                 sc = wr - 2
        wr = BC[12:7]       (14,30)             wr = wr - 2
        osr = x                                 osr = x
        cnt = x                                 cnt = x

        {mul3}                                  {mul4}
        st = 0                                  st = 0
        en = -1             (31)                en = en
        sc = x                                  sc = x
        wr = x                                  wr = x
        osr = x                                 osr = x
        cnt = x                                 cnt = x

        {}                                      {mul6}
        st = x                                  st = 16
        en = x                                  en = 31
        sc = x                                  sc = 16
        wr = x                                  wr = x
        osr = x                                 osr = x
        cnt = x                                 cnt = x

        {}
        st = x
        en = x
        sc = x
        wr = x
        osr = x
        cnt = x

        {divw1}                                 {divw2}
divw    st = 0                                  st = 0
        en = 31                                 en = -1 mod sz  (15)
        sc = wr             (16)                sc = 16
        wr = BC[12:7]       (16)                wr = wr - 1
        osr = x                                 osr = x
        cnt = x                                 cnt = x

        {divw3}                                 {divw4}
        st = wr             (16)                st = 0
        en = -1             (31)                en = 31
        sc = wr             (16)                sc = wr
        wr = BC[12:7]       (16)                wr = x
        osr = x                                 osr = x
        cnt = x                                 cnt = x

        {divw5}                                 {divw6}
        st = 4                                  st = 16
```

```
        en = -1 mod size (7)
        sc = 28
        wr = x
        osr = x
        cnt = x

        {divw7}
        st = st
        en = -1   (31)
        sc = 0
        wr = x
        osr = x
        cnt = x

        {div11}
div1    st = wr - 1      (31)
        en = -1          (31)
        sc = x
        wr = BC[12:7]    (32)
        osr = x
        cnt = x

        {div13}
        st = 0
        en = -1          (31)
        sc = 0
        wr = x
        osr = x
        cnt = x

        {}
        st = x
        en = x
        sc = x
        wr = x
        osr = x
        cnt = x

        {}
        st = x
        en = x
        sc = x
        wr = x
        osr = x
        cnt = x

        {}
unk     st = x
        en = x
        sc = x
        wr = x
        osr = x
        cnt = x
```

```
en = 31
sc = 16
wr = x
osr = x
cnt = x

{div12}
st = 0
en = -1          (31)
sc = 0
wr = wr - 1
osr = x
cnt = x

{div14}
st = 0
en = 31
sc = 0
wr = x
osr = x
cnt = x

{div16}
st = 16
en = 31
sc = 16
wr = x
osr = x
cnt = x

{}
st = x
en = x
sc = x
wr = x
osr = x
cnt = x
```

```
{}
st = x
en = x
sc = x
wr = x
osr = x
cnt = x

{}
st = x
en = x
sc = x
wr = x
osr = x
cnt = x

{}
st = x
en = x
sc = x
wr = x
osr = x
cnt = x

{}
st = x
en = x
sc = x
wr = x
osr = x
cnt = x

{asl1}
asl  st = 0
en = osr + ~wr
sc = ~wr + 1
wr = DB [5:0] or BC[12:7] (Q)
osr = BC[5:0]   (8,16,32)
cnt = x

{asl3}
st = 0
en = osr - 1
sc = x
wr = wr
osr = x
cnt = x

{}
st = x
en = x
sc = x
wr = x
osr = x
cnt = x

{}
st = x
en = x
sc = x
wr = x
osr = x
cnt = x

{}
st = x
en = x
sc = x
wr = x
osr = x
cnt = x

{unk6}
st = 16
en = 31
sc = 16
wr = x
osr = x
cnt = x

{asl2}
st = x
en = ~(wr-1) mod sz
sc = x
wr = wr
osr = osr
cnt = x

{asl4}
st = osr + ~wr
en = -1 mod sz
sc = x
wr = wr
osr = x
cnt = x

{asl6}
st = 16
en = 31
sc = 16
wr = x
osr = x
cnt = x
```

```
        cnt = x

        {asr1}                                  {asr2}
asr     st = wr                                 st = wr - 1
        en = osr - 1                            en = (wr - 1) mod sz
        sc = wr                                 sc = x
        wr = DB [5:0] or BC[12:7] (Q)           wr = wr
        osr = BC[5:0]    (8,16,32)              osr = osr
        cnt = x                                 cnt = x

        {asr3}                                  {}
        st = osr - 1                            st = x
        en = osr - 1                            en = x
        sc = x                                  sc = x
        wr = wr                                 wr = x
        osr = osr                               osr = x
        cnt = x                                 cnt = x

        {}                                      {asr6}
        st = x                                  st = 16
        en = x                                  en = 31
        sc = x                                  sc = 16
        wr = x                                  wr = x
        osr = x                                 osr = x
        cnt = x                                 cnt = x

        {}
        st = x
        en = x
        sc = x
        wr = x
        osr = x
        cnt = x

        {rot11}                                 {rot12}
rotl    st = osr                                st = x
        en = -1          (31)                   en = ~(wr - 1) mod sz
        sc = osr                                sc = x
        wr = DB [5:0] or BC[12:7] (Q)           wr = wr
        osr = BC[5:0]    (8,16,32)              osr = osr
        cnt = x                                 cnt = x

        {rot13}                                 {}
        st = 0                                  st = x
        en = 31                                 en = x
        sc = ~(wr - 1) mod sz                   sc = x
        wr = wr                                 wr = x
        osr = osr                               osr = x
        cnt = x                                 cnt = x

        {}                                      {rot16}
        st = x                                  st = 16
        en = x                                  en = 31
        sc = x                                  sc = 16
```

```
        wr = x                                  wr = x
        osr = x                                 osr = x
        cnt = x                                 cnt = x

        {}
        st = x
        en = x
        sc = x
        wr = x
        osr = x
        cnt = x

        {rotr1}                                 {rotr2}
rotr    st = osr                                st = x
        en = -1          (31)                   en = (wr - 1) mod sz
        sc = osr                                sc = x
        wr = DB [5:0] or BC[12:7] (Q)           wr = wr
        osr = BC[5:0]    (8,16,32)              osr = osr
        cnt = x                                 cnt = x

        {rotr3}                                 {}
        st = 0                                  st = x
        en = 31                                 en = x
        sc = wr mod sz                          sc = x
        wr = wr                                 wr = x
        osr = osr                               osr = x
        cnt = x                                 cnt = x

        {}                                      {rotr6}
        st = x                                  st = 16
        en = x                                  en = 31
        sc = x                                  sc = 16
        wr = x                                  wr = x
        osr = x                                 osr = x
        cnt = x                                 cnt = x

        {}
        st = x
        en = x
        sc = x
        wr = x
        osr = x
        cnt = x

        {roxl1}                                 {roxl2}
roxl    st = 0                                  st = 0
        en = osr + ~wr   (14)                   en = (osr - wr) mod sz
        sc = -1          (31)                   sc = 0
        wr = BC[12:7]    (1)                    wr = wr
        osr = BC[5:0]    (16)                   osr = osr
        cnt = x                                 cnt = x

        {roxl3}                                 {roxl4}
        st = (~(wr-1) + 1) mod sz               st = 0
```

```
en = -1 mod sz
sc = (~(wr-1) + 1) mod sz
wr = DB [5:0] or BC[12:7] (Q)
osr = BC[5:0]    (8,16,32)
cnt = x

{rox15}
st = (~(wr-1) + 1) mod sz
en = -1 mod sz
sc = (~(wr-1) + 1) mod sz
wr = wr
osr = osr
cnt = x

{rox16}
st = 16
en = 31
sc = 16
wr = wr - 1 - osr
osr = osr
cnt = x

{rox17}
st = wr - 1
en = osr - 1
sc = 0
wr = wr
osr = osr
cnt = x

{roxr1}
st = wr
en = osr - 1
sc = wr
wr = BC[12:7]    (1)
osr = BC[5:0]    (16)
cnt = x

{roxr2}
st = 0
en = (wr - 1) mod sz
sc = 0
wr = wr
osr = osr
cnt = x

{roxr3}
st = 0
en = (wr-1) - 1
sc = (wr-1) + 24,16,0
wr = DB [5:0] or BC[12:7] (Q)
osr = BC[5:0]    (8,16,32)
cnt = x

{roxr4}
st = wr
en = osr - 1
sc = wr
wr = wr
osr = osr
cnt = x

{roxr5}
st = 0
en = (wr-1) - 1
sc = (wr-1) + 24,16,0
wr = wr
osr = osr
cnt = x

{roxr6}
st = 16
en = 31
sc = 16
wr = wr - 1 - osr
osr = osr
cnt = x

{roxr7}
st = 0
en = osr - wr
sc = 0
wr = wr
osr = osr
cnt = x
```

roxr

```
        {bfrg1}                          {bfrg2}
bfreg   st = 0                           st = 0
        en = 31                          en = wr - 1
        sc = osr + wr                    sc = 0
        wr = DB[4:0] or IRC2[4:0]        wr = wr
        osr = REGB[4:0] or IRC2[10:6]    osr = osr
        cnt = x                          cnt = x

        {bfrg3}                          {}
        st = 0                           st = x
        en = 31                          en = x
        sc = osr + wr                    sc = x
        wr = wr                          wr = x
        osr = osr                        osr = x
        cnt = x                          cnt = x

        {bfrg5}                          {bfrg6}
        st = x                           st = 16
        en = x                           en = 31
        sc = x                           sc = 16
        wr = wr                          wr = wr
        osr = x                          osr = osr
        cnt[1:0] = DB [1:0]              cnt = x

        {bfrg7}
        st = 0
        en = 31
        sc = 25
        wr = x
        osr = x
        cnt = x

        {bfmt1}                          {bfmt2}
bfmt    st = 3                           st = 00:
                                             ~(osr[2:0]+(wr-1))
        en = -1          (31)            en = (osr[2:0]+(wr-1))
                                              [4:3]:~osr[2:0]
        sc = 3                           sc = 0
        wr = DB[4:0] or IRC2[4:0]        wr = wr
        osr = REGB[4:0] or IRC2[10:6]    osr = osr
        cnt = x                          cnt = (osr[2:0]+
                                               (wr-1)) [5:3]

        {bfmt3}                          {bfmt4}
        st = 0                           st = 00:
                                             ~(osr[2:0]+(wr-1))
        en = 11:~osr[2:0]                en = -1 mod sz  (7)
        sc = 0                           sc = 0
        wr = wr                          wr = wr
        osr = osr                        osr = x
        cnt = x                          cnt = x

        {bfmt5}                          {bfmt6}
        st = x                           st = 16
```

```
en = x
sc = x
wr = x
osr = x
cnt = x

{bfmt7}
st = x
en = x
sc = x
wr = x
osr = x
cnt = x
```

```
en = 31
sc = 16
wr = wr
osr = osr
cnt = x
```

```
{bfmi1}
bfmi  st = 3

en = -1          (31)

sc = 3

wr = DB[4:0] or IRC2[4:0]
osr = REGB[4:0] or IRC2[10:6]
cnt = x
```

```
{bfmi2}
st = 00:
     ~(osr[2:0]+(wr-1))
en = (osr[2:0]+(wr-1))
     [4:3]:~osr[2:0]
sc = 00:
     ~(osr[2:0]+(wr-1))
wr = wr
osr = osr
cnt = (osr[2:0]+
      (wr-1)) [5:3]
```

```
{bfmi3}
st = 0

en = 11:~osr[2:0]
sc = 11:~(osr[2:0]+(wr-1))

wr = wr
osr = osr
cnt = x
```

```
{bfmi4}
st = 00:
     ~(osr[2:0]+(wr-1))
en = -1 mod sz  (7)
sc = 00:
     ~(osr[2:0]+(wr-1))
wr = wr
osr = x
cnt = x
```

```
{bfmi5}
st = 0
en = 00:(osr[2:0]+(wr-1))
sc = 25+(00:
      (osr[2:0]+(wr-1)))
wr = wr
osr = x
cnt[1:0] = DB [1:0]

{bfmi7}
st = 0
en = 31
sc = 25
wr = x
osr = x
cnt = x
```

```
{bfmi6}
st = 16
en = 31
sc = 16

wr = wr
osr = osr
cnt = x
```

```
        {cop1}                          {cop2}
cop     st = x                          st = x
        en = x                          en = x
        sc = x                          sc = x
        wr = x                          wr = wr - 1
        osr = x                         osr = x
        cnt = x                         cnt = x

        {cop3}                          {cop4}
        st = x                          st = x
        en = x                          en = x
        sc = x                          sc = x
        wr = x                          wr = x
        osr = x                         osr = x
        cnt = x                         cnt = x

        {cop5}                          {cop6}
        st = x                          st = 16
        en = x                          en = 31
        sc = x                          sc = 16
        wr = DB [7:2]                   wr = x
        osr = x                         osr = x
        cnt[1:0] = DB [1:0]             cnt = x

        {cop7}
        st = x
        en = x
        sc = x
        wr = x
        osr = x
        cnt = x
```

1 loaded based on ird[5] - if ird[5] = 0 then wr value comes from BC bus else value is loaded from regc.

FTU - FIELD TRANSLATION UNIT OPERATIONS:

3- LDCR load the control register from regb. The register is selected by the value in ar[1:0], this can be gated onto the rx bus.

4- DPSW load the psw with the value in regb. Either the ccr or the psw is loaded depending upon size. If size = byte then only load the ccr portion.

14- CLRFP clear the f-trace pending latch. (fpend2 only)

17- LDSH2 load the contents of the shifter control registers from regb. These include wr,osr,count.

19- LDSWB load the internal bus register from regb. This is composed of bus controller state information which must be accessed by the user in fault situations.

21- LDSWI load the first word of sswi (internal status word) from regb. This is composed of tpend, fpend1, fpend2, ar latch

23- LDSH1 load the contents of the shifter control registers from regb. These include st,en,sc.

25- LDUPC load micro pc into A4 from regb and check validity of rev #.

26- LDPER load per with the value on the a-bus. (should be a T3 load). ab>per

28- LDARL load the ar latch from regb. May be able to share with ldswi or ldswj

29- ØPSWM clear the psw master bit.

33- RPER load output of per into ar latch and onto bc bus. There are two operations which use this function, MOVEM and BFFFO. MOVEM requires the least significant bit of the lower word (16-bits only) that is a one to be encoded and latched into the AR latch and onto the BC BUS (inverted) so that it can be used to point at a register. If no bits are one then the end signal should be active which is routed to the branch pla. After doing the encoding, the least significant bit should be cleared.

For BFFFO it is necessary to find the most significant bit of a long word that is a one. This value is encoded into 6 bits where the most significant bit is the 32-bit all zero signal. Thus the following bits would yield the corresponding encoding.

most sig bit set	per out	onto bc bus
31	Ø 11111	1110 0000
16	Ø 10000	1110 1111
Ø	Ø 00000	1111 1111
NONE	1 11111	0000 0000

The output is then gated onto the BC bus where it is sign extended to an 8-bit

value. It does not hurt anything in the BFFFO case to load the other latch (i.e. BFFFO can load the AR latch).
For BFFFO it does not matter if a bit is cleared.

34- STCR store the control register in regb. The register is selected by the value in ar[1:0], this can be gated onto the rx bus.

37- STPSW store the psw or the ccr in regb based on size. If size = byte then store ccr only with bits 8 - 15 as zeros.

38- 0PEND store the psw in regb then set the supervisor bit and clear the trace bit in the psw. Tpend and Fpend are cleared. The whole psw is stored in regb.

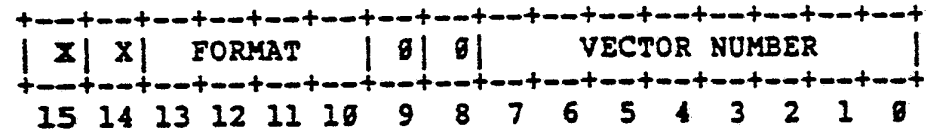
39- 1PSWS store the psw in regb then set the supervisor bit and clear both trace bits in the psw. The whole psw is stored in regb.

40- STINST store IRD decoded information onto the BC bus and into regb. This data can be latched from the BC bus into other latches (i.e. wr & osr) by other control.

41- STIRD store the ird in regb.

43- STINL store the new interrupt level in pswi and regb. The three bits are loaded into the corresponding pswi bits. The same three bits are loaded onto bc bus [3:1] with bc bus [31:4] = 1 and [0] = 1, which is loaded into regb. Clear IPEND the following T1.

44- STV# store the format & vector number associated with the exception in regb.



47- STCRC store the contents of the CRC register in regb. Latch A4 with microaddress.

48 STSH2 store the contents of the shifter control registers into regb. These include wr,osr,count. Store high portion of shift control

50- STSWB store the internal bus register in regb.

This is composed of bus controller state information which must be accessed by the user in fault situations.

52- STSWI store sswi (internal status word) in regb. The sswi is composed of tpend, ar latch, fpend1, fpend2

54- STSH1 store the contents of the shifter control registers into regb. These include st,en,sc.

56- STUPC store the micro pc in regb.

REV NUMBER	CRC	MICRO PC
15 14 13 12	11 10	9 8 7 6 5 4 3 2 1 0

62- NONE

63- STPER store the per onto the a-bus. (should be a T1 transfer). per>ab

PC - PC SECTION OPERATIONS:

	AOBP[1] 0	AOBP[1] 1
31 - 3PFI 30 - 3PFF	EV3FI TPF	OD3FI EV3FI

0- NF
```
aobpt>db>sas
tp2>ab>sas
```

1- TPF
```
aobpt>db>tp1
aobpt>db>aup>aobp*,aobpt
+2>aup
tp1>tp2
tp2>ab>sas
```

2- PCR
```
tp2>ab>a-sect
(if ry=pc then connect pc and address section)
aobpt>db>sas
```

3- PCRF
```
aobpt>db>tp1
aobpt>db>aup>aobp*,aobpt
+2>aup
tp1>tp2
tp2>ab>a-sect
```

```
        (if ry=pc then connect pc and address section)

4- JMP1
        tp2>db>a-sect
        a-sect>ab>aobpt

5- BOB
        aobpt>db>tp1
        tp1>tp2
        tp2>ab>sas

 - EV3FI
        aobpt>db>tp1*
        aobpt>db>aup>aobpt
        +4>aup
        tp2>ab>sas

 - OD3FI
        aobpt>db>aup>aobpt,tp2
        +2>aup
        tp2>ab>sas

7- TRAP
        tp2>db>a-sect
        pc>ab>sas

8- TRAP2
        tp2>ab>a-sect
        aobpt>db>sas

9- JMP2
        a-sect>ab>aobpt
        aobpt>db>sas

10- PCOUT
        pc>ab>a-sect
        aobpt>db>sas

11- NPC   Conditional update based on cc=t/f
        tp2>db>aup,a-sect
        a-sect>ab>aup>aobpt

12- LDTP2
        a-sect>ab>tp2
        aobpt>db>sas

13- SAVE1
        pad>aobp
        aobpt>db>sas
        tp2>ab>sas

15- SAVE2
        aobp>db>tp1
        tp2>ab>sas
```

```
14- FIX
        aobpt>db>tp1
        tp2>ab>aobpt
        tp1>tp2

16- LDPC
        tp2>pc
        aobpt>db>sas
        tp2>ab>sas

PIPE - PIPE OPERATIONS:

 Description of bit encodings.
     [6] = use irc
     [5] = change of flow
     [4] = fetch instruction
     [3:0] = previously defined pipe control
             functionality.
```

	AOBP[1]	
	0	1
0 1 1 3 - 3UDI	EV3Fa	OD3F
1 0 1 7 - 3UDF	TUD	EV3Fb

```
  - EV3Fa
        chr1>irb
        chrh>pb>imh,iml,irc
        change of flow
        fetch instr

  - EV3Fb
        chr1>irb
        chrh>pb>imh,iml,irc
        irc>ir                    ! implies use irc
        use pipe
        fetch instr

  - OD3F
        chr1>pb>irc
        ! force miss regardless of whether odd or even
        change of flow
        fetch instr

 0 0 0 0 - NUD
        x

 1 0 0 0 - JPIPE
        use pipe

 0 0 1 1- FIX2              Always transfer irb up pipe
```

```
        chr>irb                 to irc,im and if irb needs
        irb>pb>imh,iml,irc      to be replaced, do access
                                and transfer chr to irb.
        ! force miss regardless of whether odd or even
        change of flow,
        fetch instr

        db>ird                  else load irb from d-bus.
        irb>pb>imh,iml,irc
        change of flow
        fetch instr

0 0 0 2 - IRAD
        ira>db

0 0 0 4 - IRTOD
        ir>ird

0 0 1 5 - FIX1
        chr>irb                 if irc needs to be replaced,
                                do access and transfer chr
                                to irb, else no activity.
        ! force miss regardless of whether odd or even
        change of flow
        fetch instr

1 0 0 6 - 2TOC
        irc2>irc
        irc>ir
        use pipe

0 0 0 8 - CLRA
        clear irc2[14]
        ira>ab                          zxtd 8 -> 32

0 0 0 9 - STIRA
        db>>ira
        ira>pb>irc2

0 0 0 11 - ATOC
        db>>ira
        ira>pb>irc

0 0 1 13 - LUD
        chr>irb
        irb>pb>imh,iml
        fetch instr

1 0 0 14 - CTOD
        irc>ir,ird
        irb>irc
        use pipe

1 0 1 15 - TUD
```

```
        chr>irb
        irb>pb>imh,iml,irc
        irc>ir
        use pipe
        fetch instr

0 1 1 15 - TOAD
        chr>irb
        irb>pb>imh,iml,irc
        irc>ir
        change of flow
        fetch instr
```

MICROINSTRUCTION SEQUENCES

```
CALLM EA
A2
+---+---------+-----+-------------+-----+
|FA5| CMD01   | 36e |             | DB  |
+---+-+-----+--+---+-+----+--------+-----+
|   X|DATA|    DT|    RY| .             |
+-----+----+------+------+---------------+-----+
| "PT @ NEW PC"                          | ADD |
| AOB>DB>AU>AOB                          | NIL |
| 4>AU                                   |  X  |
| "STORE EVAL EA"                        |  X  |
| AOB>DB>>REGA                           | SBM2|
| "STORE MMU DESCRIPTOR ADDRESS"         | NONE|
| REGB>AB>AT,DT                          |  NF |
| "ST=24,EN=31,SC=24"                    | NUD |
| "CLEAR BAD IRB"                        |     |
| AOB>DB>IRB                             |3/31 |
+---+---------+-----+-------------+-----+
|FB5| CMD02   | 36f |             | DB  |
+---+-+-----+--+---+-+----+--------+-----+
| LONG|DATA|    RX|    RY| <>  T0  UNK   |
+-----+----+------+------+---------------+-----+
| "READ NEW PC"                          | ADD |
| "PT @ MDP"                             | AND |
| AOB>DB>AU>AOB                          |  X  |
| 4>AU                                   |  X  |
| "TEST TYPE"                            |  X  |
| DBIN>DB>>IRA,REGB                      |NONE |
| "BUILD 0 IN REGA"                      |  NF |
| 0>ALU>REGA                             |UATOC|
| "STORE DESC. ADDR"                     |     |
| REGA>AB>AUT                            |2/25 |
+---+---------+-----+-------------+-----+
|FC5| CMD03   | 3c0 |             | BC  |
+---+-+-----+--+---+-+----+--------+-----+
| LONG|DATA|    SP|    RY| <>  T0  UNK   |
+-----+----+------+------+---------------+-----+
| "READ MDP"                             | SUB |
| "STORE NEW PC"                         | NIL |
| DBIN>AB>AOBPT                          |  X  |
| "PT @ STACK SP"                        | INS |
| SP>DB>AU>AOB                           |  X  |
| 4>AU                                   |NONE |
| "BUILD TYPE:0:0:0"                     |JMP2 |
| % REGB>DB>SAD                          | NUD |
|                                        |     |
|                                        |12/16|
+----------------------------------------+-----+
 FD5          TYPE0^OPT0 -> CMD05   (FF5)
              TYPE0^OPT4 -> CMD04   (FE5)
                   TYPE1 -> CMD13   (FE6)
          ILLEGALFORMAT -> XFEA     (FE4)
```

FE5	CMD04	69f		DB
LONG	INST	SP	RY	<W> TØ
"WRITE STACK OLD SP"				X
SP>AB>>DOB				NIL
% AUT>DB>SA				X
				X
				X
				NONE
				NF
				NUD
				11/Ø

FF5	CMDØ5	49f		DB
BYTE	DATA	SP	RY	.
"PT @ STACK OLD PC"				SUB
SP>DB>AU>AOB,SP				NIL
ØC>AU				X
"STORE OLD PC"				X
TP2>AB>AT				X
"STORE PSW"				STPSW
% REGA>AB>SAD				TRAP2
% REGB>DB>SAD				NUD
				11/Ø

FG5	CMDØ6	3c1		DB
LONG	DATA	SP	RY	<W> TØ
"WRITE STACK OLD PC"				SUB
AT>AB>>DOB				NIL
"PT @ STACK DESCRIPTOR"				X
AOB>DB>AU>AOB,SP				X
4>AU				X
% REGB>DB>SAD				NONE
				TPF
				TOAD
				11/Ø

FH5	CMD07	3c2		DB
LONG	INST	RX	RY	<W> T0
"WRITE STACK DESCRIPTOR"				ADD
AUT>AB>>DOB				NIL
"PT @ STACK MDP"				X
AOB>DB>AU>AOB				X
8>AU				X
% REGB>DB>SAD				NONE
				TPF
				TUD
				11/0

FI5	CMD08	3c3		DB
X	DATA	RZ	RY	SPC X SYNC
"STORE OLD MDP"				X
RZ>AB>>DOB				NIL
"LOAD NEW MDP"				X
DBIN>DB>RZ				X
"SYNC TO ENSURE NO STACK"				X
"ON MDP"				NONE
				NF
				NUD
				11/0

FJ5	CMD09	3c8		DB
LONG	INST	SP	RY	<W> T0
"WRITE STACK MDP"				SUB
"PT @ STACK ARG CNT"				NIL
AOB>DB>AU>AOB,SP				X
0C>AU				X
% REGA>AB>SA				X
% REGB>DB>SAD				NONE
				TPF
				TUD
				11/0

```
+---+---------+-----+-------------+-----+
|FK5| CMD10   | 3c9 |             | DB  |
+---+-+----+--+---+-+----+--------+-----+
| BYTE|DATA|    RX|    RY| .             |
+-----+----+------+------+---------------+-----+
| "BUILD TYPE:OPL:0:ARG CNT"               |   X |
| PER>AB>ALU>REGA                          | AND |
| -1>ALU                                   |   X |
| REGA>DB>FOOLIT                           |   X |
| % AT>DB>SAA                              |   X |
|                                          |STPER|
|                                          | TPF |
|                                          | TUD |
|                                          |     |
|                                          |11/0 |
+---+---------+-----+-------------+-----+
|FL5| CMD11   | 3ca |             | DB  |
+---+-+----+--+---+-+----+--------+-----+
| WORD|DATA|    SP|    RY| <W> T0        |
+-----+----+------+------+---------------+-----+
| "WRITE 0:ARG CNT"                        | SUB |
| REGA>DB>>DOB                             | AND |
| "BUILD TYPE:0:CCR/TYPE:OPL:PSW"          |   X |
| REGB>AB>ALU>ALUT                         |   X |
| REGA>DB>FOOLIT                           |   X |
| -1>ALU                                   |NONE |
| "PT @ STACK TYPE:OPL:0:CCR"              |  NF |
| AOB>DB>AU>AOB,SP                         | NUD |
| 4>AU                                     |     |
| % AUT>AB>SAA                             |11/0 |
+---+---------+-----+-------------+-----+
|FM5| CMD12   | 3cb |             | A1  |
+---+-+----+--+---+-+----+--------+-----+
| LONG|INST|    RX|    RY| <W> T0        |
+-----+----+------+------+---------------+-----+
| "WRITE TYPE:OPL:PSW"                     |   X |
| ALUT>DB>>DOB                             | NIL |
| % AT>DB>SAA                              |   X |
| % AUT>AB>SA                              |   X |
|                                          |COL1 |
|                                          |STINS|
|                                          |  NF |
|                                          | NUD |
|                                          |     |
|                                          |11/0 |
+------------------------------------------+-----+
```

INVALID FORMAT (CMD03)

FE4	XFEA	49e		DB
X	INST	RX	RY	EXL X FORE
"STORE REAL PC"				X
PC>AB>AUT				NIL
"PSW>REGB,1>PSWS,0>PSWT"				X
"0>TPEND"				X
% AT>DB>SA				X
% REGA>AB>SAD				0PEND
				PCOUT
				NUD

FF4	XFEA2	36d		DB
X	INST	RX	RY	.
"CORRECT REAL PC"				SUB
AUT>DB>AU>AUT				NIL
2>AU				X
% REGA>AB>SA				X
% REGB>DB>SAD				X
				NONE
				NF
				NUD
				12/20

FG4 TRAP2 (IJ5)

TYPE = 1 (CMD03)

FE6	CMD13	79f		DB
IRSZ	DATA	RX	RY	.
"PT @ DESC. NPL"				ADD
AUT>DB>AU>AOB				COL2
1>AU				X
"TEST ARG CNT"				X
PER>AB>ALUT				X
"FORM STACK PROBE"				STPER
PER>AB>ALU>REGB				NF
-1>ALU				NUD
REGA>DB>FOOLIT				12/16

FF6	CMD14	481			BC
IRSZ	INST	SP	DTY	<> TØ	UNK
"READ DESC. NPL"					X
"STORE NEW MDP"					AND
DBIN>AB>ALUT					X
"BUILD MMU BASE ADDRESS"					X
Ø>ALU>DTY					SBM2
AT>DB>FOOLIT					NONE
"STORE STACK PTR"					NF
SP>AB>>AOB					NUD
					1/11

FG6 LOCZ -> CMD18 (FH6)
LOCZ -> CMD15 (FH7)

ARG CNT <> Ø - MAKE STACK PROBE (CMD14)

FH7	CMD15	551	BFME3 (EK5)		DB
WORD	DATA	RX	RY	.	
"PT @ STACK PROBE"					ADDXS
AOB>AB>AU>AOB					NIL
REGB>DB>AU					X
% REGA>AB>SAD					X
					X
					NONE
					NF
					NUD
					1/11

FI7	CMD16	4a7			DB
IRSZ	DATA	DT	RY	<> TØ	
"MAKE STACK PROBE"					X
"PT @ MMU CPL"					NIL
DT>DB>AOB					X
"STORE NPL"					X
DBIN>AB>>REGB					X
% AUT>AB>SAA					NONE
					NF
					NUD
					12/16

FJ7	CMD17	52e	CMD37 (FS8)	DB
IRSZ	DATA	RX	RY	<> T0	CPU1
"READ MMU CPL"				X
"PT @ MMU DESC"				NIL
AT>DB>>AOB				X
% AUT>AB>SA				X
% REGB>DB>SAD				X
				NONE
				NF
				NUD
				1/11

FK7 CMD20 (FJ6)

ARG CNT = 0 - NO STACK PROBE (CMD14)

FH6	CMD18	550	FERI1 (AF2)	DB
X	DATA	RX	DTY	.	
"PT @ MMU CPL"				X
DTY>DB>>AOB				NIL
% REGA>AB>SA				X
				X
				X
				NONE
				NF
				NUD
				1/11

FI6	CMD19	483		DB
IRSZ	DATA	RX	RY	<> T0	CPU1
"READ MMU CPL"				X
"PT @ MMU DESCRIPTOR"				NIL
AT>DB>AOB				X
"STORE NPL"				X
DBIN>DB>>REGB				X
% AUT>AB>SA				NONE
				NF
				NUD
				12/16

FJ6	CMD20	489			DB
LONG	DATA	DT	RY	<W> T0	CPU1
"WRITE DESCRIPTOR ADDRESS"					ADD
AUT>AB>>DOB					NIL
"PT @ MMU IPL"					X
DT>DB>AU>AOB					X
8>AU					X
					NONE
					NF
					NUD
					12/16

FK6	CMD21	48b			DB
IRSZ	DATA	RX	RY	<W> T0	CPU1
"WRITE NPL TO IPL"					SUB
REGB>DB>>DOB					NIL
"PT @ MMU STATUS"					X
AOB>DB>AU>AOB					X
4>AU					X
% AUT>AB>SA					NONE
					NF
					NUD
					12/16

FL6	CMD22	48d			DB
IRSZ	INST	SP	DTY	<> T0	CPU1
"READ MMU STATUS"					X
"BUILD TYPE:CPL:0:0"					NIL
DBIN>DB>>REGB					X
"SHUFFLE NEW MDP"					INS
ALUT>AB>DTY					X
"STORE OLD SP"					NONE
SP>DB>AT					NF
% AUT>AB>SAA					NUD
					12/16

```
|FM6| CMD23    | 4a1 |              | DB   |
+---+-+----+--+---+-+----+--------+------+
| IRSZ|INST|     RX|     RY| .             |
+-----+----+------+------+--------+------+
| "STORE MMU STATUS"                |    X|
| DBIN>DB>REGC                      |  NIL|
| DBIN>DB>ALUT                      |    X|
| "SHUFFLE ARG CNT"                 |    X|
| PER>AB>>REGB                      | SBM5|
|                                   |STPER|
|                                   |   NF|
|                                   |  NUD|
|                                   |     |
|                                   |12/16|
+---+---------+-----+-------------+------+
|FN6| CMD24    | 4a3 |             | BC   |
+---+-+----+--+---+-+----+--------+------+
| IRSZ|DATA|     SP|     RY| .             |
+-----+----+------+------+--------+------+
| "PT @ STACK SP"                   |  SUB|
| SP>DB>AU>AOB                      |  NIL|
| 4>AU                              |    X|
| "STORE OLD SP"                    |    X|
| AT>AB>>DOB                        |    X|
| "STORE ARG CNT"                   |STPSW|
| REGB>DB>>IRA                      |   NF|
| "Ø:CCR IN REGB"                   |STIRA|
|                                   |     |
|                                   |12/21|
+-----------------------------------+------+
 FO6                 WR<>Ø -> CMD26   (FP6)
              WR=Ø^CNT<>Ø -> CMD27   (FP5)
               WR=Ø^CNT=Ø -> XFEA    (FE4)
```

VALID - SP CHANGE (CMD24)

```
+---+---------+-----+-------------+------+
|FP6| CMD26    | 59f |             | BC   |
+---+-+----+--+---+-+----+--------+------+
| LONG|INST|     RX|     RY| <>  T3  UNK   |
+-----+----+------+------+--------+------+
| "READ NEW SP"                     |  ADD|
| AUT>DB>AU>AOB                     |  NIL|
| ØC>AU                             |    X|
| "STORE DESCRIPTOR ADDRESS"        |    X|
| AUT>DB>ALUT                       |    X|
| % REGA>AB>SA                      | NONE|
|                                   |   NF|
|                                   |  NUD|
|                                   |     |
|                                   |12/16|
+-----------------------------------+------+
 FQ6                 OPT4 -> CMD29   (FR5)
                     OPTØ -> CMD3Ø   (FR6)
```

VALID - NO SP CHANGE (CMD24)

FP5	CMD27	59e		DB
LONG	DATA	SP	DTY	<W> T0
"WRITE STACK SP"				SUB
"PT @ STACK OLD PC"				NIL
AOB>DB>AU>AOB,SP				X
8>AU				X
"STORE OLD PC"				X
TP2>AB>AT				NONE
"POSITION NEW MDP"				TRAP2
DTY>AB>DBIN				NUD
% REGB>DB>SAD				

FQ5 CMD06 (FG5)

NO STACK COPY (CMD26,CMD28)

FR5	CMD29	69c		DB
X	DATA	RX	RY	.
"PT @ STACK SP"				SUB
DBIN>DB>AU>AOB				NIL
4>AU				X
% AUT>AB>SA				X
				X
				NONE
				NF
				NUD

FS5 CMD27 (FP5)

STACK COPY (CMD26)

FR6	CMD30	49c		DB
	X | INST | SP | RY | .			
"STORE OLD SP"				X
SP>AB>>AOB,REGB				NIL
SP>AB>AUT				X
"SHUFFLE ARG CNT"				X
IRA>DB>REGC				SBM5
				NONE
				NF
				IRAD
				1/05
FS6	CMD31	4a5		BC
	X | DATA | SP | RY | .			
"PT @ TOP OF NEW STACK"				SUB
DBIN>DB>AU>AT,SP				NIL
IRA>AB>AU				X
"IRA ZXTD 8->32"				X
				SBM6
				NONE
				NF
				CLRA
				1/05

FT6 WR<>0 -> CMD35 (FS7)
WR=0^CNT<>0 -> CMD37 (FS8)
WR=0^CNT=0 -> CMD39 (FS9)

TYPE 1 - COPY LONG (CMD31,CMD36)

FS7	CMD35	529	4-WAY SHARE	DB
LONG | DATA | RX | RY | <> T0				
"READ OLD STACK"				ADD
"PT @ NEW STACK"				NIL
AT>DB>>AOB				X
"PT @ NEXT NEW STACK ENTRY"				X
AT>DB>AU>AT				X
4>AU				NONE
% REGA>AB>SA				NF
% REGB>DB>SAD				NUD
				1/05

FT7	CMD36	4a9		BC
LONG	DATA	RX	RY <W> T0	
"WRITE TO NEW SP"				ADD
DBIN>DB>>DOB				NIL
"PT @ NEXT OLD STACK ENTRY"				X
AUT>DB>AU>AOB,AUT				X
4>AU				SBM6
% AT>AB>SA				NONE
				NF
				NUD
				1/05

FU7 WR<>0 -> CMD35 (FS7)
WR=0^CNT<>0 -> CMD37 (FS8)
WR=0^CNT=0 -> CMD39 (FS9)

TYPE 1 - COPY LAST PIECE (CMD31,CMD36)

FS8	CMD37	528	4-WAY SHARE	DB
SSIZE	DATA	RX	RY <> T0	
"READ OLD STACK"				X
"PT @ LAST NEW STACK"				NIL
AT>DB>>AOB				X
% AUT>AB>SA				X
% REGB>DB>SAD				X
				NONE
				NF
				NUD
				1/05

FT8	CMD38	4f0	2-WAY SHARE	DB
SSIZE	INST	RX	RY <W> T0	
"WRITE NEW STACK"				X
DBIN>DB>>DOB				NIL
% AT>DB>SAA				X
% AUT>AB>SA				X
				X
				NONE
				NF
				NUD
				1/05

FU8 CMD39 (FS9)

```
 TYPE 1 - NONE LEFT TO COPY        (CMD31,C
+---+---------+-----+-------------+-----+
|FS9| CMD39   | 428 | 4-WAY SHARE | DB  |
+---+-+----+--+---+-+----+--------+-----+
|    X|INST|    RX|  VDT1| .              |
+-----+----+------+------+--------+-------+
| "PT @ TOP OF OLD STACK"                 |  X  |
| % ALUT>DB>AT                            | NIL |
| % AUT>AB>VDT1                           |  X  |
|                                         |  X  |
|                                         |  X  |
|                                         |NONE |
|                                         | NF  |
|                                         | NUD |
|                                         |     |
|                                         |1/05 |
+---+---------+-----+-------------+-----+
|FT9| CMD40   | 4b5 |             | DB  |
+---+-+----+--+---+-+----+--------+-----+
| BYTE|DATA|    SP|    RY| .              |
+-----+----+------+------+--------+-------+
| "PT @ STACK SP"                         | SUB |
| SP>DB>AU>AOB                            | NIL |
| 4>AU                                    |  X  |
| "SHUFFLE DESCRIPTOR ADDRESS"            |  X  |
| ALUT>AB>AUT                             |  X  |
| "STORE OLD SP"                          |STPSW|
| REGB>DB>DOB                             | NF  |
|                                         | NUD |
|                                         |     |
|                                         |1/10 |
+-----------------------------------------+-----+
 FU9       CMD27    (FP5)
```

```
 RTM RY
 A1
+---+---------+-----+-------------+-----+
|FA8| RTM01   | 4aa |             | DB  |
+---+-+----+--+---+-+----+--------+-----+
| LONG|INST|    SP|   DTY| <>  T1         |
+-----+----+------+------+--------+-------+
| "READ TYPE:OPL:PSW"                     |  X  |
| SP>DB>>AOB                              | NIL |
| "STORE DESCRIPTOR ADDRESS"              |  X  |
| REGB>DB>DTY                             |  X  |
| "ST=24,EN=31,SC=24"                     |SBM2 |
| % AUT>AB>SA                             |NONE |
|                                         | NF  |
|                                         | NUD |
|                                         |     |
|                                         |3/31 |
+---+---------+-----+-------------+-----+
```

FB8	RTM02	4ab		DB
BYTE	DATA	DT	RY	.
"PT @ STACK ARG CNT"				ADD
AOB>DB>AU>AOB				AND
4>AU				X
"FORM MMU BASE ADDRESS"				X
Ø>ALU>DT				X
REGB>DB>FOOLIT				NONE
% AUT>AB>SA				NF
				NUD

FC8	RTMØ3	4ad		DB
WORD	DATA	RX	RY	<> TØ
"READ STACK ARG CNT"				ADD
"PT @ STACK PC"				NIL
AOB>DB>AU>AOB				X
8>AU				ZXTD
"STORE TYPE:OPL:PSW"				SBM2
DBIN>DB>>REGA				NONE
"ST=16,EN=23,SC=16"				NF
% AUT>AB>SA				NUD
				12/Ø4

FD8	RTM04	4ae		DB
LONG	DATA	RX	RY	<> TØ
"READ STACK PC"				ADD
"PT @ STACK MDP"				NIL
AOB>DB>AU>AOB				X
4>AU				ZXTD
"STORE ARG CNT"				X
DBIN>AB>AT				NONE
"LOAD TYPE FOR BRANCH"				NF
REGB>DB>>IRA				UATOC
				2/25

```
|FE8| RTM05   | 4af |              | BC   |
+---+-+----+--+---+-+----+--------+------+
| LONG|DATA|    RX|    RY| <>  T0        |
+-----+----+------+------+--------+------+
| "READ STACK MDP"                 | ADD  |
| "PT @ STACK SP"                  | NIL  |
| AOB>DB>AU>AOB                    |  X   |
| 4>AU                             |  X   |
| "STORE NEW PC"                   |  X   |
| DBIN>AB>AOBPT                    | NONE |
| "SHUFFLE OPL"                    | JMP2 |
| REGB>DB>>DOB                     | NUD  |
|                                  |      |
|                                  |12/21 |
+----------------------------------+------+
 FF8              TYPE0 -> RTM06   (FG8)
                  TYPE1 -> RTM09   (FG9)
          ILLEGALFORMAT -> XFEB    (FE7)
```

```
 TYPE 0          (RTM05)
 VALID TYPE 1    (RTM13)
+---+---------+-----+------------+------+
|FG8| RTM06   | 48f |            | DB   |
+---+-+----+--+---+-+----+--------+------+
| BYTE|INST|    RX|    RY| .             |
+-----+----+------+------+--------+------+
| "PT @ SP AFTER STRIP"            | ADD  |
| AOB>DB>AU>AOB                    | NIL  |
| 4>AU                             |  X   |
| "RESTORE MDP REG"                |  X   |
| DBIN>AB>RY                       |  X   |
| "STORE NEW PSW"                  |LDPSW |
| REGA>DB>>REGB                    |  NF  |
|                                  | NUD  |
|                                  |      |
|                                  |      |
+----------------------------------+------+
```

```
+---+---------+-----+-------------+-----+
|FH8| RTM07   | 62a |JMP1    (ES2)| DB  |
+---+-+-----+--+---+-+----+--------+-----+
|   X|INST|    SP|    RY| .              |
+-----+----+------+------+--------+------+
| "PT @ FINAL STACK VALUE"        |ADDX6|
| AOB>AB>AU>SP                    | NIL |
| AT>DB>AU                        |   X |
| % REGA>AB>SAD                   |   X |
| % REGB>DB>SAD                   |   X |
|                                 |NONE |
|                                 |3PFI |
|                                 |3UDI |
|                                 |     |
|                                 |1/08 |
+---------------------------------+-----+
 FI8       DBCC5    (EU7)
```

```
 TYPE = 1            (RTM05)
+---+---------+-----+-------------+-----+
|FG9| RTM09   | 78f |             | DB  |
+---+-+-----+--+---+-+----+--------+-----+
| LONG|DATA|    DT|    RY| <>  T0         |
+-----+----+------+------+--------+------+
| "READ STACK SP"                 | ADD |
| "PT @ MMU DPL"                  | NIL |
| DT>DB>AU>AOB                    |   X |
| 0C>AU                           |   X |
| "STORE NEW MDP"                 |   X |
| DBIN>AB>>REGB                   |NONE |
| % AUT>AB>SAA                    |  NF |
|                                 | NUD |
|                                 |     |
|                                 |     |
+---+---------+-----+-------------+-----+
|FH9| RTM10   | 4b1 |             | DB  |
+---+-+-----+--+---+-+----+--------+-----+
| BYTE|DATA|    DT|    RY| <W> T0  CPU1   |
+-----+----+------+------+--------+------+
| "WRITE OLP TO DPL"              | SUB |
| "STORE NEW SP"                  | NIL |
| DBIN>AB>DT                      |   X |
| "PT @ MMU STATUS"               |   X |
| AOB>DB>AU>AOB                   |   X |
| 8>AU                            |NONE |
| % AUT>AB>SAA                    |  NF |
| % REGB>DB>SAD                   | NUD |
|                                 |     |
|                                 |     |
+---+---------+-----+-------------+-----+
```

```
|FI9| RTM11   | 4b2 |            | DB   |
+---+-+----+--+---+-+----+-------+------+
| BYTE|INST|    RX|    RY| <>  TØ  CPU1 |
+-----+----+------+------+-------+------+
| "READ STATUS"                  |    X |
| % AT>DB>SA                     |  NIL |
| % AUT>AB>SA                    |    X |
|                                |    X |
|                                |    X |
|                                | NONE |
|                                |   NF |
|                                |  NUD |
|                                |      |
+---+---------+-----+------------+------+
|FJ9| RTM12   | 4b3 |            | DB   |
+---+-+----+--+---+-+----+-------+------+
| BYTE|INST|    RX|    RY| .             |
+-----+----+------+------+-------+------+
| "TEST STATUS"                  |    X |
| DBIN>DB>ALUT                   |  NIL |
| % AT>DB>SAA                    |    X |
| % AUT>AB>SA                    |    X |
|                                |    X |
|                                | NONE |
|                                |   NF |
|                                |  NUD |
|                                |      |
+---+---------+-----+------------+------+
|FK9| RTM13   | 4b4 |            | BC   |
+---+-+----+--+---+-+----+-------+------+
|    X|INST|    DT|    RY| .             |
+-----+----+------+------+-------+------+
| "PT @ STACK SP"                |  SUB |
| DT>DB>AU>AOB                   |  NIL |
| 4>AU                           |    X |
| "SHUFFLE NEW MDP"              |    X |
| REGB>AB>DBIN                   |    X |
| % AUT>AB>SAA                   | NONE |
|                                |   NF |
|                                |  NUD |
|                                |      |
|                                |12/21 |
+--------------------------------+------+
 FL9                LOCZ -> XFEB    (FE7)
                    LOCZ -> RTMØ6   (FG8)
```

```
INVALID TYPE   (RTM05)
INVALID STATUS (RTM13)
+---+---------+-----+-------------+-----+
|FE7| XFEB    | 48e |             | DB  |
+---+-+-----+-+---+-+-----+-------+-----+
|   X|INST|   RX|   RY| EXL X   FORE |
+----+----+-----+-----+-------+-----+
| "BACK UP PC"                  | SUB|
| TP2>DB>AU>AUT                 | NIL|
| 2>AU                          |   X|
| "PSW>REGB,1>PSWS,0>PSWT"      |   X|
| "0>TPEND"                     |   X|
| % AUT>AB>SA                   |0PEND|
| % REGB>DB>SAD                 | TRAP|
|                               |  NUD|
|                               |     |
|                               |12/21|
+-------------------------------+-----+
 FF7      TRAP2   (IJ5)
```

Claims

1. A data processor adapted to cooperate with an access controller to control access to a module stored in a storage device, the data processor comprising:

first means for receiving an instruction which requests access to said module, said instruction specifying an address within said storage device containing an access request;

second means for retrieving said access request from said storage device;

third means for providing said access request to said access controller;

fourth means for allowing said requested access to said module unless said access request is denied by said access controller.

2. The data processor of claim 1 further comprising:

fifth means for vectoring to an exception handler if said access request is denied by said access controller.

3. The data processor of claim 1 wherein the module specified by said instruction is a code module, said instruction also specifing a selected number of arguments to be passed to said code module, and wherein said fourth means passes said arguments to said code module before allowing said requested access.

4. The data processor of claim 1 wherein said fourth means comprise:

fifth means for receiving a decision from said access controller to said access request; and

sixth means for allowing said requested access to said module in response to an affirmative decision from said access controller, and denying said requested access to said module in response to a negative decision from said access controller.

5. The data processor of claim 4 further comprising:
seventh means for vectoring to an exception handler if said access request is denied by said access controller.

6. In a data processor adapted to cooperate with an access controller to control access to a module stored in a storage device, a method comprising the steps of:
receiving an instruction which requests access to said module, said instruction specifying an address within said storage device containing an access request;
retrieving said access request from said storage device;
providing said access request to said access controller;
allowing said requested access to said module unless said access request is denied by said access controller.

7. In the data processor of claim 6, the method comprising the further step of:
vectoring to an exception handler if said access request is denied by said access controller.

8. In the data processor of claim 6 wherein the module specified by said instruction is a code module, said instruction also specifing a selected number of arguments to be passed to said code module, the step of allowing said access further comprising passing said arguments to said code module before allowing said requested access.

9. In the data processor of claim 6, the step of allowing said access comprising the steps of:
receiving a decision from said access controller to said access request; and
allowing said requested access to said module in response to an affirmative decision from said access controller, and denying said requested access to said module in response to a negative decision from said access controller.

10. In the data processor of claim 9, the method comprising the further step of:

vectoring to an exception handler if said access request is denied by said access controller.

11. In a data processor adapted to cooperate with an access controller to control access to a module stored in a storage device, a method comprising the steps of:

receiving an instruction which requests access to said module, said instruction specifying an address within said storage device containing an access request;

retrieving said access request from said storage device;

providing said access request to said access controller;

receiving a decision from said access controller to said access request; and

allowing access to said module in response to an affirmative decision from said access controller, and denying access to said module in response to a negative decision from said access controller.

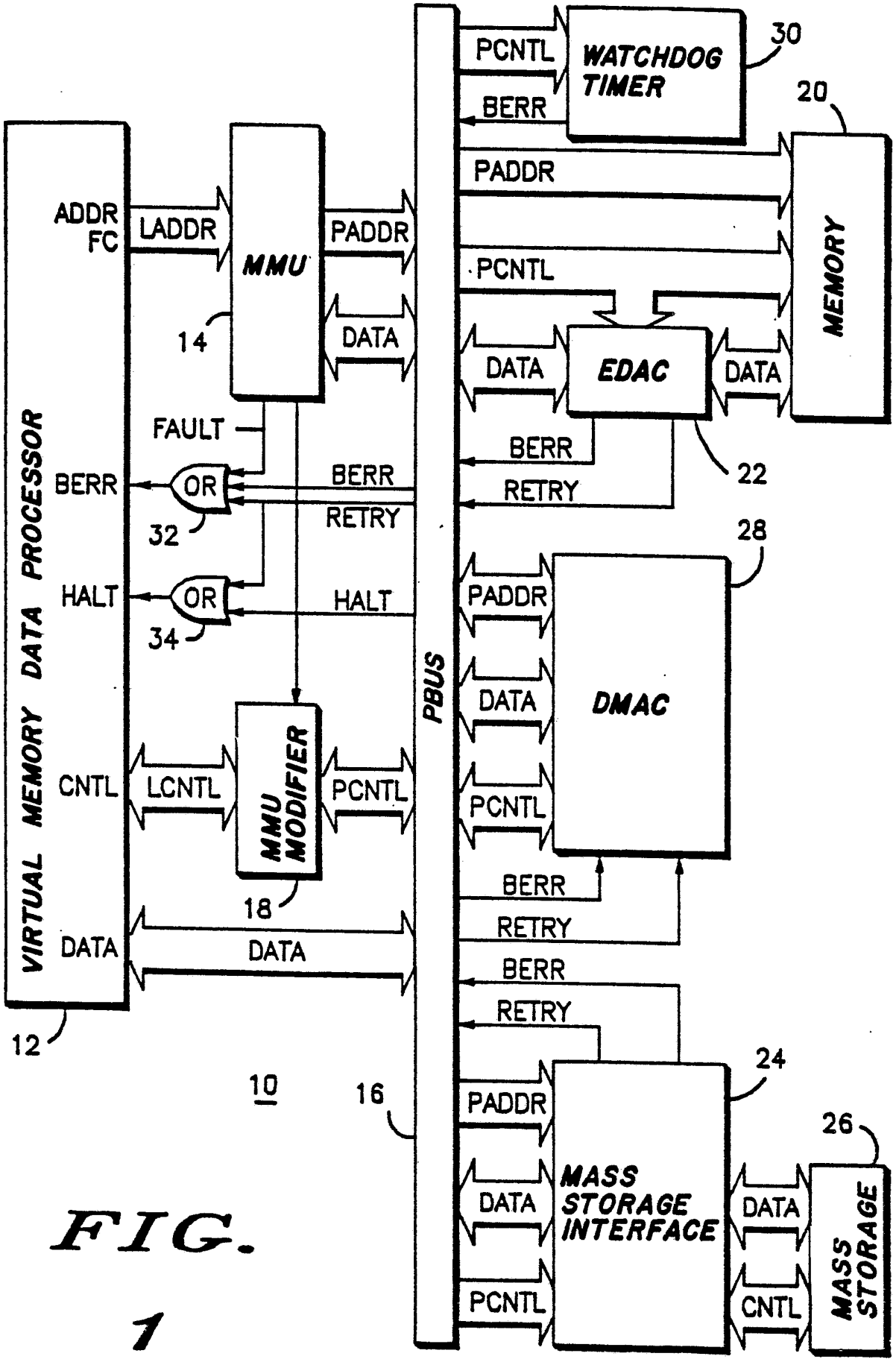

VIRTUAL MEMORY DATA PROCESSOR
ADDR
FC
LADDR
MMU
PADDR
DATA
14
FAULT
BERR
OR
32
HALT
OR
34
CNTL
LCNTL
MMU MODIFIER
PCNTL
18
DATA
DATA
12
10
16
PBUS
PCNTL
WATCHDOG TIMER
30
BERR
20
PADDR
PCNTL
MEMORY
DATA
EDAC
DATA
BERR
RETRY
22
BERR
RETRY
HALT
28
PADDR
DATA
DMAC
PCNTL
BERR
RETRY
BERR
RETRY
24
PADDR
DATA
MASS STORAGE INTERFACE
PCNTL
26
DATA
CNTL
MASS STORAGE

FIG. 1

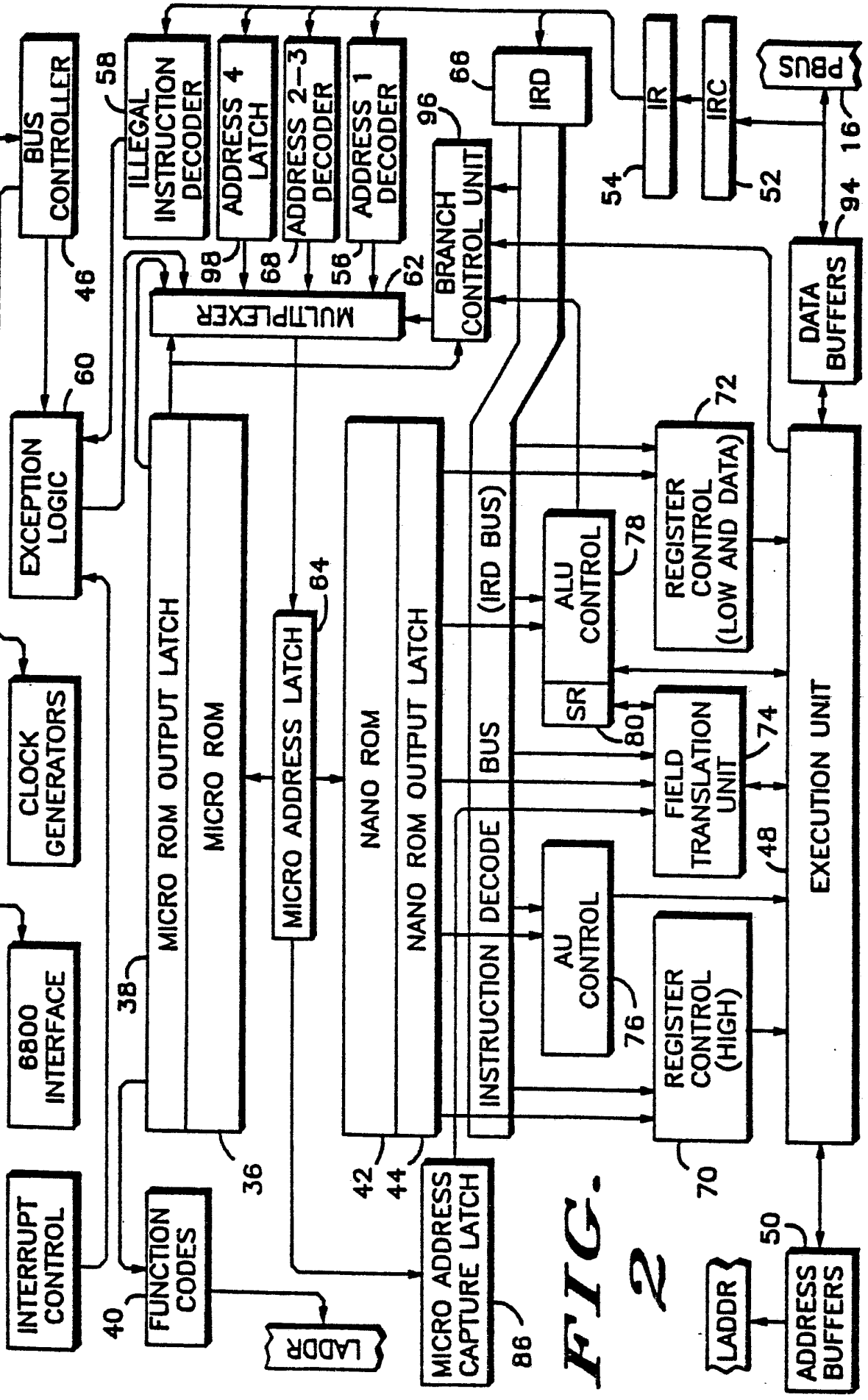

BUS CONTROLLER
ILLEGAL INSTRUCTION DECODER
ADDRESS 4 LATCH
ADDRESS 2-3 DECODER
ADDRESS 1 DECODER
IRD
IR
IRC
PBUS
MULTIPLEXER
BRANCH CONTROL UNIT
DATA BUFFERS
EXCEPTION LOGIC
CLOCK GENERATORS
6800 INTERFACE
INTERRUPT CONTROL
FUNCTION CODES
LADDR
MICRO ROM OUTPUT LATCH
MICRO ROM
MICRO ADDRESS LATCH
NANO ROM
NANO ROM OUTPUT LATCH
MICRO ADDRESS CAPTURE LATCH
(IRD BUS)
BUS
DECODE
INSTRUCTION
ALU CONTROL
SR
AU CONTROL
REGISTER CONTROL (LOW AND DATA)
FIELD TRANSLATION UNIT
REGISTER CONTROL (HIGH)
EXECUTION UNIT
LADDR
ADDRESS BUFFERS
46
58
98
68
56
62
96
68
54
52
94
16
60
72
78
80
74
48
38
64
36
42
44
70
50
40
86

FIG. 2

INTERNATIONAL SEARCH REPORT

International Application No PCT/US85/00735

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all) [3]

According to International Patent Classification (IPC) or to both National Classification and IPC 3

INT. CL. G 06F 9/00, G 06F 9/46
U.S. CL. 364/200

II. FIELDS SEARCHED

Minimum Documentation Searched [4]

Classification System	Classification Symbols
US	364/200 MS File 364/900 MS File

Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched [5]

III. DOCUMENTS CONSIDERED TO BE RELEVANT [14]

Category *	Citation of Document, [16] with indication, where appropriate, of the relevant passages [17]	Relevant to Claim No. [18]
Y	US, 4,434,464 28 February 1984 Suzuki et al	1-11
Y	US, 4,104,721 1 August 1978 Markstein et al	1,3-5,7-9, 11-13,15-17&19
A	US, 4,442,484 10 April 1984 Childs, Jr. et al	
A	US, 4,366,537 28 December 1982 Heller et al	
A	US, 4,177,510 4 December 1979, Appell et al	
A P	US, 4,488,228 11 December 1984 Crudele et al	
Y	US, 4,183,085 8 January 1980 Roberts et al	1-11

* Special categories of cited documents: [15]

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

IV. CERTIFICATION

Date of the Actual Completion of the International Search [2]	Date of Mailing of this International Search Report [2]
6 JUNE 1985	26 JUN 1985
International Searching Authority [1]	Signature of Authorized Officer [20]
ISA/US	

Form PCT/ISA/210 (second sheet) (October 1981)