Control:

S: Sets virtual DKP busy to one and clears DKP done and the end of cylinder error flag in the virtual DKP status word. If the command in the virtual command and track register is recalibrate, a recalibration flag is set and the action corresponding to a seek is performed (except clearing the recalibrate flag). If the command is a seek, the recalibrate flag is cleared and the seek done flag for the selected drive is set in the virtual DKP status word. If drive 0 is selected, the virtual seek error flag is set if the cylinder address is out of range. Should drive 1 or 2 be selected, the error flag is set if the track address is greater than 63. The DKP busy flag is cleared and the done flag is set to one. If virtual interrupts are enabled for DKP, a virtual interrupt is pending.

If the command is a read or write, all but the data channel mapping portion of a disk record is immediately generated. The record contents are:

1. Identification: disk I/O / current virtual processor / last record.

2. Drive: drive field of virtual DASCR.

3. Disk Address.

4. Cylinder: If the recalibrate flag is set, then the physical track or cylinder corresponding to logical track (cylinder) zero; otherwise, if drive is zero, then cylinder field of command and cylinder register plus the virtual processor base cylinder address; otherwise, the track field of the virtual command and track register.

5. Surface: if drive is zero, then the surface field of the virtual DASCR.

6. Starting Sector: if drive is zero, then the starting sector field of the virtual DASCR MOD 12, otherwise, the starting sector field MOD 8.

7. Sector Count: if drive is zero, then 16 MINUS the sector count field of the virtual DASCR; otherwise (16 MINUS the sector count field) MOD 9.

8. Memory Address: contents of virtual memory address register.

9. Read/Write: contents of command field of virtual command and cylinder register.

The VMM now determines which portions of the virtual store need to be realized prior to the virtual disk transfer, realizes these pages, and defines an appropriate data channel mapping for the virtual disk I/O.

The address in the virtual memory address register corresponds to a virtual processor logical address only if virtual data channel mapping is enabled. This is indicated by bit one in the virtual MMPU status word. Should virtual data channel mapping be disabled, the address is physical. Whether logical or physical, the starting and stopping page addresses for the virtual disk transfer may be computed by

starting page address = virtual memory address register DIV 1024

stopping page address = [virtual memory address register PLUS (disk record sector count TIMES 256)] DIV 1024.

Assuming these are virtual processor physical pages, the page descriptor for each page indicates if it is realized. If so, a disk record mapping field may be loaded corresponding to the page descriptor. If so, the page fault handler is entered. Upon completion of the page fault handling routine, the updated page descriptor is used to load a disk record mapping field.

If the page fault handler is to be entered, the frames of all realized pages needed in the virtual disk transfer should be entered on the locked list to prevent their replacement. The previous fault instruction should be set to the current instruction to prevent clearing of the locked list on first entry. Whenever, a frame is swapped in, its frame will also be placed on the locked list to prevent its replacement.

The VMM can determine if the entire contents of a page is to be overwritten by a virtual disk read instruction. In this case, if the page is not realized, the page fault handler does not need to swap in the page after it has obtained a free frame. To do so would be needless since the entire contents of the page will be lost due to the virtual disk transfer.

If the starting memory address corresponds to a virtual processor logical address, then the virtual processor physical address of each page is determined from the virtual data channel map A. If the virtual map indicates a validity violation for a page, this information is loaded into the disk record mapping field. Execution of the disk record by the disk driver will result in a data channel error. This condition is detected by the disk process and a virtual data channel error is simulated. Similarly, if the virtual data channel map indicates write protection and the virtual write protect enable flag is set in the virtual MMPU status word, this information should be loaded into the disk record mapping

field.

If the data channel map contains a virtual processor physical page address, the page descriptor is used to determine if it is realized. If so, the disk record mapping field is loaded with the double mapping. Otherwise, the page fault handler is entered. On completion, the updated page descriptor is used to load the double mapping information.

When the disk record is finally completed, the ready bit of the virtual DKP status word is cleared and the record is sent to the disk queuer.

C: The virtual DKP busy flag, done flag, all virtual error flags, and all virtual seek done flags are cleared. Virtual disk transfers in progress continue.

P: Same action as S except that the virtual busy flag is not altered and all virtual error flags are cleared. A read or write operation initiated with a P while the virtual busy flag is zero will not request a virtual interrupt upon I/O completion. This command is normally issued to initiate a seek.

3.3.7 CDR (Card Reader) Instructions. -

A virtual processor may perform I/O on the virtual card reader only if it is mapped to the real card reader. Virtual card readers are equivalent to real card readers in all respects except for data loss potential. In a real card reader, once a card enters the read station, characters are read every 400 microseconds until the end of card is reached. Characters must be processed at this rate to prevent data loss. Because a program executing on a virtual processor cannot be expected to respond to character input within 400 microseconds, the VMM must buffer the entire card. Hence, virtual card readers do not lose data.

When a virtual processor executes an instruction to pick a card and read the first character, the VMM reads the entire card and stores the contents in a buffer. Subsequent character reads by the virtual processor only require the VMM to fetch the next character from the buffer.

Components of the virtual store referenced by instructions addressing CDR are:

1. virtual accumulators

2. virtual CDR status word

3. virtual CDR busy flag

4. virtual CDR done flag

The VMM also maintains an eighty character input buffer, a buffer pointer, and a flag which is set when buffering occurs. The component of the buffer selected by the pointer corresponds to the virtual CDR input buffer.

The following actions correspond to transfer and control commands.

Transfer:

DIA (read column): if the VMM is currently buffering a card or if the buffer has been read, then the designated virtual accumulator is loaded with the last component of the character buffer. Otherwise, the buffer pointer is incremented and the contents of the selected component are loaded into the designated virtual accumulator.

DIB (read status): If the VMM is currently buffering a card then the ready bit of the virtual CDR status word is cleared and the status word is loaded into the designated virtual accumulator. Otherwise, the real CDR status word is loaded into both the virtual CDR status word and the designated virtual accumulator.

Control:

S: The virtual CDR busy flag is set to one and the done flag is cleared. The buffering flag is set, the buffer pointer is reinitialized, and an attempt is made to read the first character from the real card reader. If the read succeeds, it is loaded into the character buffer after the buffer pointer is incremented. Should the read fail, the buffering indicator is cleared.

C: Clear the virtual CDR busy and done flag.

P: If the end of buffer has been reached, the virtual CDR busy flag is cleared. In any case, virtual done is set to one. If interrupts are enabled for this device, an interrupt request is pending.

3.3.8 LPT (Line Printer) Instructions. -

A virtual processor may perform I/O on the virtual line printer only if it is mapped to the real line printer. Components of the virtual store referenced by instructions addressing LPT are:

1. virtual LPT busy flag
2. virtual LPT done flag
3. virtual LPT character buffer
4. virtual LPT status word

The virtual character buffer and status word for this device are directly supported by real line printer registers. The following actions correspond to transfer and control commands:

Transfer:

DOA (load character buffer): the real device character buffer is loaded with the contents of the designated virtual accumulator.

DIA (read status): the designated virtual accumulator is loaded with the contents of the real device status register.

Control:

S: The virtual LPT busy flag is set to one and the virtual done flag is cleared. A real start command is issued to the real line printer.

C: The virtual LPT busy and done flags are cleared. A real clear command is issued to the real device.

3.3.9 TTI1 And TTO1 (Second Teletype) Instructions. -

A virtual processor may perform I/O on the virtual second teletype only if it is mapped to the real teletype (devices TTO and TTI). Components of the virtual store referenced instructions addressing TTI1 are:

1. virtual TTI1 busy flag
2. virtual TTI1 done flag

3. virtual TTI1 character buffer

The virtual TTI1 character buffer is directly supported by the character buffer of the real TTI device.

The following actions correspond to transfer and control commands addressing TTI1:

Transfer:

DIA (read character buffer): The designated virtual accumulator is loaded with the real TTI character buffer.

Control:

S: The virtual TTI1 busy flag is set to one and the virtual done flag is cleared. A real start instruction is issued to the real TTI device.

C: The virtual TTI1 busy and done flags are cleared. A real clear instruction is issued to the real TTI device.

Components of the virtual store referenced by instructions addressing TTO1 are:

1. virtual TTO1 busy flag

2. virtual TTO1 done flag

3. virtual TTO1 character buffer

The virtual TTO1 character buffer is directly supported by the character buffer of the real TTO device.

The following actions correspond to transfer and control commands addressing TTO1:

Transfer:

DOA (load character buffer): The real TTO device character buffer is loaded with the contents of the designated virtual accumulator.

Control:

S: The virtual TTO1 busy flag is set to one and the virtual done flag is cleared. A real start command is issued to the real TTO device.

C: The virtual TTO1 busy and done flags are cleared. A real start command is issued to the real TTO device.

3.4 Auto Increment/Decrement Instruction Interpretation.

If an auto increment or auto decrement instruction is attempted while a virtual processor is assigned the real processor, the trap process will be awakened. It must first determine if a virtual auto location violation has occurred; such is the case if the virtual MMPU status word indicates that virtual auto protection is enabled. If a virtual violation is not present, the instruction must be interpreted by the VMM. The auto increment/decrement facility is supported if the virtual processor is in supervisor mode (case 1) or if it is in user mode with virtual logical page zero mapped to virtual physical page zero (case 2). If virtual logical page zero is mapped elsewhere (case 3), interpretation is required but the auto facility is not supported.

Interpretation for case one proceeds as follows. Determine the effective data address of the trapping instruction (range 20 to 37 octal) and fetch the contents of this location from the virtual store. If the data address was in the range 20 to 27 (octal), increment the contents just fetched; otherwise decrement them. Store the updated contents back into the realized virtual store. If these contents specify an indirect address, fetch the word they point to. This process ends when the last address in the chain is fetched. Should any link or the final word pointed to not be realized, the page fault handler is entered. The final address fetched is used in the software interpretation of the instruction.

In case two, addresses generated are virtual processor logical addresses. Interpretation is the same as case one except that virtual program maps are used to obtain virtual processor physical addresses. If a virtual logical page is found to be validity protected, a virtual validity violation is simulated. Similarly, if instruction interpretation requires writing on a virtual logical page that is virtual write protected, a virtual write violation is simulated.

Case three is the same as two except that the initial auto increment or decrement is not performed. In all cases, the only instructions which are candidates for interpretation are the memory reference instructions (LDA, STA, JMP, JSR, ISZ, and DSZ).

3.5 Virtual Protection Facility.

The virtual protection features of a virtual processor are enabled only when the virtual processor is in virtual user mode and protection enable flags in the virtual MMPU status word are set to one. Any protection violation which occurs while the virtual processor is assigned the real processor awakes the trap process. If the trap process determines that virtual protection is enabled, then a virtual

protection violation is supported. Virtual violation simulation begins by fetching the address of the virtual violation handler from virtual physical location 47. If this is an indirect address, the chain is followed until the end. Should links not be realized, the page fault handler is entered. The address of the virtual violation handler is loaded into the virtual program counter, the virtual logical address of the violating instruction is placed in the virtual violation address register, the inhibit flag (bit 2) of the virtual MMPU status word is set to one, and the virtual MMPU and CPU busy flags are cleared. The virtual violation data register is loaded with the virtual logical page address of the violation and appropriate violation flags are set.

The trap process determines virtual violations in the following manner:

1. Virtual validity violation occurs when a virtual logical page is mapped to virtual physical page 127 (octal) and the write and validity protect flags of the virtual map register are set.

2. Virtual write violation occurs when a virtual logical page is write protected and the write protect enable flag in the virtual MMPU status word is set to one.

3. Virtual auto location violation occurs when the auto location protect enable flag of the virtual MMPU status word is set to one.

4. Virtual I/O violation occurs when the I/O protect enable flag of the virtual MMPU status word is set to one.

5. Virtual defer violation occurs when the defer protect enable flag of the virtual MMPU status word is set to one.

In the real processor, the following violation combinations are possible.

auto : defer
auto : validity
defer : validity
auto : defer : validity

If the trap process detects any of these virtual violations, it also checks for the other two to ensure that the virtual violation data register is correct.

Explicit traps are simulated in the same fashion except that the address of the trapping instruction is placed in virtual physical location 46 and the virtual violation data register is not altered.

3.6 Entry And Exit In The Trap Process.

On entry to the trap process, the state of the processor at the time of the trap is saved. State includes accumulators, carry bit, stack and frame pointers, and program counter. During the trap process, components of the virtual store may be altered.

On exit, either the real processor is reassigned to the virtual processor or the dispatcher process is awakened. If the preempt condition is true initially, the dispatcher process is awakened. Otherwise, if a virtual interrupt or stack fault is pending, it is simulated. Should simulation result in a virtual mode switch or page fault, the virtual processor is preempted and the dispatcher process is awakened. Otherwise, the virtual processor regains the real processor.

CHAPTER FOUR

The Dispatcher, Stack Fault, and Device Processes

4.1 The Dispatcher Process.

The dispatcher process is awakened whenever a virtual processor is preempted. Should no virtual processor be in a ready state, the dispatcher holds the processor and awaits an interrupt. Interrupts schedule device processes which may promote virtual processors to the ready state. In this case, return to the dispatcher results in selection of a ready virtual processor and its promotion to running.

Whenever more than one virtual processor is ready to run, the VMM must determine which virtual processor is to be selected and how long it is to run. These considerations characterize the scheduling policy. It is desirable to provide a mechanism through which a wide range of policies may be implemented according to specified parameters. To this end, the dispatcher process provides a round robin variable quantum service discipline. Quanta for the virtual processors are specified by the operator and may be changed at any time. Should a quantum be set to zero, the virtual processor will not be preempted until a page fault or explicit fault occurs. This scheduling mechanism allows policies ranging from equal quantum round robin to priority round robin to first come first serve.

Once a virtual processor is selected, promotion to running proceeds in several steps. First, virtual processor state is checked for the existence of a pending virtual interrupt or stack fault. If one exists, it is simulated. Should a page fault result, the virtual processor is put in the blocked state and another virtual processor is selected.

The next step is composition of a real address translation map. If the virtual processor is in virtual supervisor mode, a real address map is created based on the frame table. On the other hand, if virtual user mode is present, virtual memory management must be supported. The real address map is based on the virtual program map selected in the virtual MMPU status word and on the page descriptors.

Promotion to running is completed by initializing the alarm clock with the specified quantum, realizing the virtual accumulators, carry bit, stack pointer, and frame pointer, and assigning the real processor to the virtual processor.

4.2 The Stack Fault Process.

The stack fault process is awakened by a real processor stack fault. Should another VMM process be executing, the real processor stack pointer is inspected for possible overflow into virtual processor address space. If so, the condition is communicated to the operator.

Should a virtual processor be assigned the real processor when the stack fault occurs, a virtual stack fault pending condition is set by the VMM. On exit from the stack fault process, the dispatcher process is awakened only if the end of the current time slice has been reached. Normally, the stack fault is simulated. Should a virtual mode switch or page fault occur during simulation, the dispatcher process is awakened. If not, the virtual processor regains the real processor.

4.3 The QTY (Multiplexer) Process.

The multiplexer process is awakened by a real machine interrupt from the multiplexer device. CRTs connected to the multiplexer are potentially mapped to virtual processors, which consider them to be TTI and TTO devices. The multiplexer process first determines if the interrupt corresponds to successful transmission or reception of a character and which CRT line is involved. In the case of transmission, the virtual busy flag of the appropriate virtual TTO is cleared and the virtual TTO done flag is set to one. If the virtual interrupt disable flag for the virtual TTO device is zero, a virtual interrupt is pending.

In the case of a reception, the character received is transferred to the appropriate virtual TTI character buffer. Virtual TTI busy is cleared and TTI done is set to one. A virtual interrupt is pending if the virtual interrupt disable flag of the virtual TTI device is set.

In both cases, the VMM clears the real machine interrupt condition.

4.4 The RTC (Real Time Clock) Process.

The real time clock process is awakened by a real time clock interrupt at the rate of one thousand times per second. It serves three functions. First, a thirty-two bit system clock is maintained which increments every time the process is awakened. This clock overflows approximately one and a half months after the monitor is started. Second, it simulates virtual real time clocks by decrementing the virtual clock counter of the running virtual processor. Should the counter become zero, it is reset according to the contents of the virtual RTC output buffer. If the virtual RTC busy flag is equal to one, it is cleared and and virtual RTC done is set to one. A virtual interrupt for this device is pending if its virtual interrupt disable flag is zero.

Finally, an alarm clock counter is maintained which times a virtual processors time slice. This clock is initialized by the dispatcher process and decrements as long as it is nonzero. When it becomes zero, the running virtual processor is preempted.

4.5 The DKP (Disk) Process

The disk process is awakened by an interrupt from the real DKP device, signifying the completion of either a seek or read/write disk operation. The disk driver is called to continue the disk operation according to the current disk record, or to signal completion of a disk transfer. Transfer completion requires further action. The disk queuer is called to schedule another disk record if one is waiting and the used record is returned to the record manager.

Further action is required if the last record of a page fault, bootstrap load, or virtual disk I/O sequence is encountered. When a page fault is serviced, the state of the corresponding virtual processor is assigned ready. At the end of the bootstrap load sequence, a message is sent to the operator verifying this condition. Finally, at the end of virtual disk I/O, the contents of the real disc address and sector count register, status register, and memory address register are copied into their virtual counterparts. If the virtual DKP busy flag is equal to one, it is cleared and virtual DKP done is set to one. A virtual interrupt for the virtual DKP device is pending if its interrupt disable flag is equal to zero.

4.6 The CDR (Card Reader) Process.

The card reader process is awakened by an interrupt from the real card reader. Interrupts from this device occur only when the VMM is reading a card for a virtual device. The character just read is stored in the selected component of the character buffer after the buffer pointer is incremented. If the end of card is reached, the buffering indicator is cleared, the buffer pointer is reset, virtual CDR busy is cleared, and virtual CDR done is set to one. A virtual interrupt is pending for the virtual CDR device if its interrupt disable flag is set to zero. If the end of card is not reached, an instruction is issued to the real card reader to set up an interrupt when the next character is read.

4.7 The LPT (Line Printer) Process.

The line printer process is awakened by an interrupt from the real line printer. Virtual LPT busy is cleared and virtual LPT done is set to one. A virtual interrupt is pending for this device if the virtual LPT disable flag is set to zero.

4.8 The TTO And TTI (Teletype) Processes.

These processes are awakened by interrupts from the real TTO and TTI devices. These real devices are potentially mapped to the virtual second teletype (TTO1 and TTI1) of a virtual processor. For each device, if virtual busy is cleared and virtual done is set to one. A

virtual interrupt is pending if the corresponding virtual device (TTO1 or TTI1) interrupt disable flag is zero.

4.9 The TTO1 And TTI1 (Second Teletype) Processes.

These processes are awakened by interrupts from the real TTO1 and TTI1 devices. These devices correspond to the real second teletype, which serves as the VMM operator's console. The three basic functions are operator console communication, system generation, and command line interpretation.

4.9.1 Operator Console Communication. -

Communication with the VMM operator's console is performed through an input and an output buffer. Typing a character on the console keyboard causes an interrupt, which awakens the TTO1 process. The character is echoed and stored in the input buffer. A carriage return specifies the end of line; the input buffer is processed as either a system generation input or a command line.

Output is performed by loading a message in the output buffer and printing the first character. At this point, the keyboard is locked to prevent character echoing in the output line. When a character is printed, an interrupt awakes the TTO1 process, which prints the next character in the buffer. When the last character is printed, the keyboard is unlocked. The TTO1 process is not awakened after the printing of an echo character.

4.9.2 System Generation. -

When the virtual machine monitor is started, several parameters need to be specified. These are the number of virtual machines desired, a frame partition size for each virtual machine, and a default quantum. As a response to each query is given, the TTI1 process executes a program which loads system tables. A response is not accepted unless it is considered to be legal. Responses are free format.

4.9.3 Command Line Interpretation. -

Once the system is generated, further inputs are considered to be command lines, of which there are three types. First, there are commands which cause simulation of virtual CPU console functions. Next, there are commands which control allocation of real devices. Finally, there are commands which serve other functions. A command line is interpreted only if it is syntactically and semantically legal. All commands are free format. Complete syntax diagrams are given in Appendix C.

4.9.3.1 Virtual Console Simulation. -

A real processor normally has a front panel which permits various functions to be performed. In order to provide these functions for virtual processors, the commands must be entered at the VMM operator's console. Keywords are used to distinguish commands. Given below is a list of available commands. For each command, its keyword, syntax, and function is given.

1. Loadswitches: LOAD <virtual processor> <number>: The virtual data switches of the designated virtual processor are loaded with the specified number.

2. Readswitches: READ <virtual processor>: The content of the virtual data switches of the designated virtual processor are printed.

3. Register Deposit: REGD <virtual processor> <register>: The contents of the virtual data switches of the designated virtual processor are loaded into the designated virtual register. Possible virtual registers are AC0, AC1, AC2, AC3 (accumulators), SP (stack pointer), FP (frame pointer), and PC (program counter).

4. Register Examine: REGE <virtual processor> <register>: The contents of the designated virtual register of the designated virtual processor are printed out.

5. Deposit into Memory: DEP <virtual processor>: The contents of the virtual data switches of the designated virtual processor are loaded into the location pointed to by the virtual program counter. Memory reference commands are interpreted only if the program counter is in the range 0..1023.

6. Deposit into Next Memory: DEPN <virtual processor>: The virtual program counter of the designated virtual processor is incremented. The contents of the virtual data switches are loaded into the location pointed to by the virtual program counter.

7. Examine Memory: EX <virtual processor>: The contents of the virtual data switches of the designated virtual processor are loaded into the the virtual program counter. The contents of the program counter and the contents of the location it points to are printed out.

8. Examine Next Memory: EXN <virtual processor>: The virtual program counter of the designated virtual processor is incremented. The contents of the virtual program counter and the contents of the location it points to are printed out.

9. Start: STAR <virtual processor>: If the designated virtual processor is terminated, the contents of the specified virtual switches are loaded into the virtual program counter and the virtual processor is promoted to ready.

10. Stop: STOP <virtual processor>: The state of the virtual processor is set to terminated. If it is currently running or interpreting, it it preempted.

11. Continue: CONT <virtual processor>: If the designated virtual processor is terminated, it is promoted to running.

12. Reset: RESE <virtual processor>: The state of the designated virtual processor is set to terminated. If it is currently running or interpreting it is preempted. All virtual busy and done flags and virtual interrupt disable flags are cleared.

4.9.3.2 Device Allocation Control. -

The operator must have some means of mapping real devices to virtual devices. Specifically, the line printer, card reader, multiplexer CRTs, second teletype, 6030 diskettes, and 4234 disk cylinders must be allocated. Three commands are used to control device allocation. These are:

1. Allocate Device: ALLO <virtual processor> <device> [<number>]: The designated device is allocated to the designated virtual processor. A device may be allocated to only one virtual processor at a time. Available devices are LPT, CRT, QTY (teletype), TTY1 (second teletype), DKP0 (4234 disk), DKP1 (6030 diskette), and DKP2 (6030 diskette). In the case of QTY allocation, the number indicates which QTY is allocated. In the case of DKP0, the number specifies the number of cylinders to be allocated. This number cannot be greater than the number of available cylinders. Once cylinders are allocated, they cannot be released, nor can the size of the allocation be changed.

2. Release Device: RELE <virtual processor> <device> [<number>]: The designated device is released from the designated virtual processor. A device may be released only if it is currently allocated to the selected virtual processor. DKP0 cylinders may not be released. In the case of QTY, the number indicates which CRT is to be released.

3. Owner of Device: OWN <device> [<number>]: Prints out the number of the virtual processor which owns the designated device or the message NOT ALLOCATED. In the case of QTY, the number indicates which CRT is selected. In the case of DKP0, the number designates a virtual processor and the number of cylinders allocated is printed out.

4.9.3.3 Other Commands. -

Several other commands serve various functions. These are:

1. Decimal: DEC: Set the input/output radix to ten.

2. Octal: OCTA: Set the input/output radix to eight.

3. Show: SHOW <number>: Print out the value of the designated show variable. These variables are counters which may be incremented at any point of program executed by a VMM process. They can indicate various kinds of information, such as the number of preemptions or page faults a virtual processor has experienced.

4. Quantum: QUAN <virtual processor>: Set the quantum of the designated virtual processor to the specified number.

5. Boot: BOOT <virtual processor> <number>: Transfer the specified number of pages from a 6030 diskette to the disk which holds the virtual store. Page zero is transferred into the appropriate frame in real main memory.

4.10 Entry And Exit From Device And Stack Fault Processes.

Upon entry to a device or stack fault process, the state of the real processor at the time of the interrupt or stack fault is saved. The program counter is recovered from real processor physical location zero. A counter is also incremented which indicates how many levels of interrupts are present.

Upon exit, several possibilities exist. If the stack fault or device process put another VMM process to sleep, it is reawakened. Otherwise, the virtual processor may need to be reassigned the real processor. First, the virtual processor is checked for the preempt condition. If it exists, the dispatcher process is awakened. If it is not preempted, the virtual store is checked for the presence of a pending stack fault or interrupt. Should it be necessary, the stack fault or interrupt is simulated. If simulation results in preemption due to virtual mode switch or a page fault, the dispatcher process is awakened. If the dispatcher is not awakened, the virtual processor regains the real processor.

CHAPTER FIVE

Implementation of the Virtual Machine Monitor

5.1 Introduction.

The structure of the virtual machine monitor has been defined in the preceeding chapters. If the design goal has been achieved, implementation should proceed in a straight forward fashion.

There are four implementation issues to be considered. First, what programming language is to be used? Second, how are processes to be scheduled? Third, what system tables and variables are needed? Finally, what programs are required to implement the functions of VMM processes?

5.2 Systems Programming Language.

The virtual machine monitor is implemented in a version of PASCAL developed for the Nova 3/D at the University of Texas at Austin. Several papers [12] [13] [14] describe the compiler and procedures for its use. The compiler translates PASCAL source code into the Nova macro assembly language, producing only innocuous instructions. Nova PASCAL is suitable for systems programming due to the following features:

1. All object code is generated in line, hence, no run time support is required.

2. Actual parameters and local variables are maintained on a stack, thus, procedures and functions may be coded in a reentrant fashion.

3. Assembly language may be embedded within PASCAL source lines, allowing the programmer to access all machine instructions.

4. Names of global variables and top level procedures and functions are used literally in code generation, permitting easy access to PASCAL defined objects by assembly language code.

Several features available in standard PASCAL are omitted in Nova PASCAL for reasons described in [14]. According to [14], omitted features are

1. floating point numbers

2. user defined scalar types

3. pointers

4. sets

5. multidimensional arrays

6. FOR, CASE, and WITH statements

7. input/output statements

8. most system functions and procedure calls

9. GOTOs and labels

Omissions (2), (5), and (6) may be overcome by suitable programming techniques. Input/output is accomplished by writing assembly language I/O routines.

In some cases, it is necessary to load the contents of a PASCAL defined variable into an accumulator before an assembly language instruction using that accumulator is issued. Similarly, it is often necessary to load a PASCAL variable with the result of an assembly language instruction. So that it does not interfere with compiler generated code, this type of assembly code is encapsulated in procedure and function bodies. In this way, the relation between generated and non generated assembly code may be well understood.

5.3 Process Scheduling.

All VMM processes are scheduled by the Nova hardware. The address of the entry program for device processes is put in location 000001 (octal). The address of the entry program for the stack fault process is put in location 000003. Finally, the address of the entry program for the trap process is put in location 000047. Whenever an interrupt, stack fault, or trap occurs, the hardware initiates execution of the appropriate entry program.

5.4 Systems Variables.

The following system variables are needed by the programs to be described.

1. currentvm: Number of currently running virtual machine.

2. numbervm: Number of virtual machines in the system.

3. clock: Alarm clock which times virtual processor time slices.

4. ltime: Low order bits of the system clock.

5. utime: High order bits of the system clock.

6. prempt: Flag which indicates that current virtual machine should be preempted.

7. ch: Character fetched from input buffer.

8. token: Type of token fetched from input buffer.

9. value: Value of token fetched from input buffer.

10. radix: Input/output radix of console communication.

11. genstate: State of system generation.

12. invmm: Flag indicating trap or dispatcher process has been awakened.

13. ilevel: Counter for stack fault/interrupt level.

14. trap: Flag indicating explicit trap instruction was executed.

15. aviol: Flag indicating auto violation occurred.

16. dviol: Flag indicating defer violation occurred.

17. iviol: Flag indicating I/O violation occurred.

18. vviol: Flag indicating validity violation occurred.

19. mviol: Flag indicating which program map was selected when violation occurred.

5.5 Systems Tables.

The following system tables are needed. These data structures are arrays.

1. istack: Stack for saving program counter after interrupt.

2. inbuff: Input buffer for console communication.

3. outbuff: Output buffer for console communication.

4. lex: Lexical table indicating type of all characters.

5. vmstate: Holds state of all virtual processors.

6. vm: Holds virtual accumulators, stack pointer, and frame pointer for each virtual processor.

7. vml: Holds the virtual processor store not in vm as well as virtual MMPU registers, TTO registers, TTI registers, RTC registers, and DKP registers.

8. vmpa: Holds virtual program map a for each virtual processor.

9. vmpb: Holds virtual program map b for each virtual processor.

10. vmda: Holds virtual data channel map a for each virtual processor.

11. vmdb: Holds virtual data channel map b for each virtual processor.

12. iostate: Holds virtual registers for the virtual CDR, LPT, TTI1, and TTO1.

13. qtyalloc: Indicates owner of each CRT.

14. qtyown: Indicates CRT owned by each virtual machine.

15. ioalloc: Indicates owner of virtual devices LPT, CDR, DKP1, DKP2, and TTY1.

16. dkp0alloc: Indicates 4234 cylinder allocation of each virtual machine.

17. show: Counters supporting SHOW function.

18. ft: Frame table.

19. locked: Locked bits.

20. dirty: Dirty bits.

21. fbase: Indicates first frame of frame partition for each virtual machine.

22. fsize: Indicates the size of the frame partition for each virtual machine.

23. falloc: Indicates the number of frames used for each virtual machine.

24. flocked: Indicates the number of frames locked for each virtual machine.

25. pd: Page descriptors for each virtual machine.

26. rw: Reserved word list.

27. rwtype: List of values for reserved word tokens.

5.6 Programs.

Programs are resources of processes; several processes may share the same program or may even execute the same program concurrently provided that it is reentrant. All shared programs of the VMM are reentrant. Because the Nova VMM is a subject of research, programs implementing it are likely to be modified. If the VMM is well designed, these changes will concern the means of implementing a function rather than the definition of the function itself. In this discussion, only a description of the function of VMM programs will be given. Program listings should be consulted for an exact implementation specification. As the majority of the code is written in PASCAL and adheres to standard conventions, it should not be difficult to follow.

5.6.1 Assembly Language Programs. -

The following code is written in assembly language.

Device process entry routine: Saves current processor state and calls main program of the device process. Depending on which device process is awakened, PASCAL defined variables may be loaded with different components of real machine state.

Stack fault process entry routine: Saves current processor state and calls main program for stack fault process. This program is the same as the device entry routine except that it has a different entry point.

Device and stack fault process exit routine: Returns processor to another stack fault or device process program, the dispatcher, or a virtual processor. Return to dispatcher requires that the state of the virtual processor be saved. Return to the virtual processor requires calling of a program to test for and simulate virtual stack faults and interrupts.

Trap process entry routine: Saves virtual processor state and loads selected MMPU registers into PASCAL defined variables. Calls the trap process main program.

Trap process exit routine: Returns processor to the dispatcher or a virtual processor. Return to virtual processor requires calling of a program to test for and simulate virtual stack faults and interrupts.

Stack initialization routine: Initializes the real processor stack and frame pointers as well as the interrupt stack. Executed by the initialization process.

5.6.2 Initialization Programs. -

The following programs are executed by the initialization process.

Procedure initl: Initializes the lexical table and the reserved word list.

Procedure enterrw: Enters words in the reserved word list.

Procedure init2: Initializes all system variables and tables.

5.6.3 Dispatcher Programs. -

The following programs are executed by the dispatcher process.

PASCAL main program (following DISP label): Selects a ready virtual processor and promotes it to running.

Procedure supermap: Composes a real address translation map in program map A using information in the frame table. Prerequisite to starting a virtual processor in virtual supervisor mode.

Procedure usermap: Composes a real address translation map in program map B using information in a virtual program map and the page descriptors. Prerequisite to starting a virtual processor in virtual user mode.

5.6.4 TTI1 Programs. -

The following programs are executed by the TTI1 process.

Procedure pttil: Main program of the TTI1 process. Handles console communication input and calls programs for command line interpretation and system generation.

Procedure legaltoken: Semantic routine which tests legality of input.

Procedure cli: Interprets command lines; calls other programs to interpret most commands.

Function showfunction: Interprets the SHOW command.

Function ownfunction: Interprets the OWN command.

Function bootfunction: Interprets the BOOT command.

Function clifunction: Interprets the virtual console commands and the RESET and ALLOCATE commands.

Procedure sysgen: Initiates the printing of system generation queries and the interpretation of inputs. This program may also be executed by the TT01 process.

5.6.5 Console Communications Programs. -

The following programs perform operation on the input and output buffers. They may be executed by the TTI1 or Trap processes.

Procedure putin: Place a character in the input buffer.

Procedure writec: Place a character in the output buffer.

Procedure writeln: Call writec to place a carriage return and line feed in the output buffer.

Procedure prompt: Call writeln; call writec to place a star in the output buffer.

Procedure writes: Call writec to place a string in the output buffer.

Procedure writen: Call writec to place a numeral in the output buffer.

Procedure sendobuff: Initiate printing of the output buffer.

Procedure nextch: Fetch a character from the input buffer.

The following program may be executed by the TT01 or Trap process.

Procedure putch: Waits for TT01 device to become idle and prints a character.

5.6.6 Syntactic Programs. -

The following programs are used to scan the input buffer and fetch tokens. They are executed by the TTI1 process.

Procedure getoken: Fetch the next token from the input buffer.

Procedure getnumb: Fetch a numeral from the input buffer and convert it to a number.

Procedure getid: Fetch an identifier from the input buffer and look it up in the reserved word list.

5.6.7 Disk Subsystem Programs. -

The following programs comprise the disk subsystem and may be executed by the TTI1 process, the trap process, and the DKP process.

Procedure diskdriver: Performs disk I/O based on a disk record.

Procedure enterqueue: Places a disk record in the disk record queue.

Procedure scheduledisk: Schedules a disk I/O if the disk is idle.

Procedure getrecord: Obtains a free disk record.

Procedure releaserecord: Frees a disk record.

Function Availrecord: Indicates if any disk records are free.

5.6.8 Page Fault Handling Programs. -

The following programs are used to handle page faults. They may be executed by any process.

Procedure faulthandler: Generates the disk records for swapping pages in and out. Updates page descriptors. Implements the replacement algorithm.

Procedure clearlocks: Clears the locked list of a virtual machine.

Procedure enterlock: Enters a frame on the locked list of a virtual machine.

Procedure setlock: Prevents clearing of locked list on execution of faulthandler.

5.6.9 Device Process Main Programs. -

The following programs are main programs for device processes.

Procedure pttol: Handles operator console output communication.

Procedure prtc: Simulates the actions of the virtual real time clocks and handles the end of a time slice. Due to performance considerations, the incrementing of the system clock and the decrementing of the alarm clock is handled by the rtc process entry routine.

Procedure pqty: Simulates the actions of virtual TTO and TTI devices.

Procedure pdkp: Initiates scheduling of the disk, simulates the actions of virtual disks, completes the handling of page faults and bootstrap loading, and continues incomplete disk I/O.

Procedure plpt: Simulates the actions of the virtual line printer.

Procedure pcdr: Simulates the actions of the virtual card reader.

Procedure ptto: Simulates the actions of the virtual TTO1 device.

Procedure ptti: Simulates the actions of the virtual TTI1 device.

5.6.10 Trap Process Programs. -

The following programs are executed by the trap process.

Procedure ptrap: Main program of the trap process. Initiates the handling of page faults detected by validity violations, interprets sensitive instructions, simulates the auto location facility, and simulates virtual traps. Other programs are called to implement some of these functions after ptrap has determined what to do.

Procedure simultrap: Simulates a virtual trap due to a virtual protection violation or a virtual explicit trap.

Procedure simulauto: Simulates the auto increment/decrement facility.

Procedure simulmri: Simulates the execution of memory reference instructions.

5.6.11 Other Programs. -

The following programs may be executed by any process.

Procedure savst: The state of a virtual processor is saved in its virtual store.

Procedure isfts: The name of this procedure stands for Interrupt Stack Fault Test Simulate. Tests for a pending interrupt or stack fault and simulates if required.

APPENDIX A

How to Run Systems Program on the Nova

Systems programs, which require a basic machine interface, may be booted in and run by means of RDOS (real time disk operating system) facilities. The steps for running a systems program are:

1. Generate an assembly language source program. This is done by actually writing assembly language, compiling a PASCAL program, or both. SPEED or MEDIT may be used to edit the program text.

2. Assemble the source program using the Nova Macro Assembler. Output from the assembler is relocatable binary. The RDOS command for assembling a program and producing an output listing on the line printer is

   MAC <filename>.SR $LPT/L

   The standard extension for assembly language files is SR. The RB extension is given to relocatable binary files.

3. Create a save file by loading the relocatable binary file. The command for loading a systems program is

   RLDR/Z/I <filename>.RB

   The Z switch is used to force loading of the save file to begin at location zero. The I switch prevents loading of routines and tables needed by programs running under RDOS. Output from the loader is a save file having the extension SV.

4. Boot the save file into core. In this operation, the core resident portion of RDOS is replaced by the save file, which must be in primary partition DP0. The command for booting in a save file is:

   BOOT DP0

   RDOS responds by requesting the name of the save file. The dialogue is:

   ```
   MASTER DEVICE RELEASED
   FILENAME? <filename>.SV
   ```

   RDOS may then say:

   ```
   PARTITION IN USE
   TYPE C TO CONTINUE
   ```

   When C is typed, the word CONTINUE is echoed and the processor dies.

It is possible that RDOS will refuse to boot in the save file and will complain that an overlay file (<filename>.OL) does not exist. This is known to occur if the save file specifies a value for location five or if no values are specified in the first 256 locations. Programs of this nature should be avoided due to this RDOS error.

5. Set the data switches to the starting address.

6. Hit the RESET and START switches.

APPENDIX B

How to Run the Virtual Machine Monitor

The following dialogue exemplifies the use of the UT virtual machine monitor. Upper case letters designate output from the computer. Lower case letters designate input by a human operator. Comments which explain the example dialogue are enclosed in slashes. The keyword of each command is identified by the first four letters of the command.

```
/ RDOS dialogue to start system at real machine first teletype (TTY) /

 boot dp0
 MASTER DEVICE RELEASED
 FILENAME ? vmm
 PARTITION IN USE - TYPE C TO CONTINUE
 CONTINUE

/ set data switches to 000002 , hit RESET, hit RUN; the following
dialogue occurs on the real machine second teletype (CRT) /

 U.T. AUSTIN   NOVA 3/D VIRTUAL MACHINE MONITOR   VERSION 1.0
 SYSTEM GENERATION - ENTER DECIMAL VALUES
 NUMBER OF VIRTUAL MACHINES ? 4

/ four virtual machines are created /

 QUANTUM (MS) ? 50
 FRAME ALLOCATION
 40 FRAMES LEFT
 ALLOCATION FOR VM 0 ? 15
 25 FRAMES LEFT
 ALLOCATION FOR VM 1 ? 10
 15 FRAMES LEFT
 ALLOCATION FOR VM 2 ? 8
 7 FRAMES LEFT
 ALLOCATION FOR VM 3 ? 7

/ system generation is completed; the monitor now prompts the operator /

 *boot 0 15

/ the virtual store of vm 0 is booted into the system; a message signals
completion /

 BOOT COMPLETE
 *boot 1 10
 BOOT COMPLETE

/ device allocation /

 *allo 0 qty 0
 *allo 1 qty 1
```

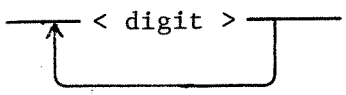
```
*allo 5 qty 2          / a reference to a non existing vm is an error /
ERROR
*allo 0 dkp0 100       / allocate 100 4234 cylinders to vm 0 /
*own qty 0
0
*own qty 1
1
*own qty 2
NOT ALLOC
*own dkp0 0
100
*own dkp0 1
0
*allo 1 lpt
*own lpt
1
*release 1 lpt
*own lpt
NOT ALLOC

/ the following commands will start vm 0 at location 64 /

*load 0 000064
*read 0
000064
*octal
*read 0
000100
*reset
*status 0
TERMINATED
*start 0
*status 0
RUNNING
*stop 0
*status 0
TERMINATED

/ the registers of vm 0 are examined /

*rege 0 ac0
000000
*rege 0 ac1
000001
*rege 0 sp
000200

/ registers may be altered /

*load 0 000007
*regd 0 ac0
*rege 0 ac0
000007

/ the VMM is terminated by hitting the STOP switch on the front panel /
```

APPENDIX C

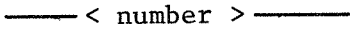
Syntax Diagrams for the VMM Command Language

In these diagrams, nonterminals are enclosed in angle brackets and terminals are set in upper case.

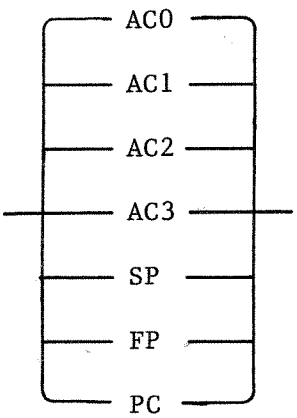
< number >



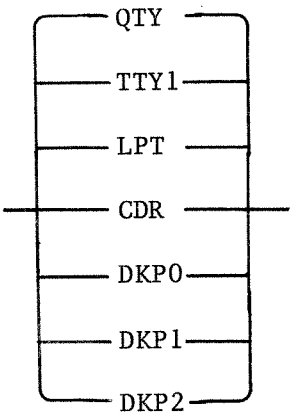
< vp >



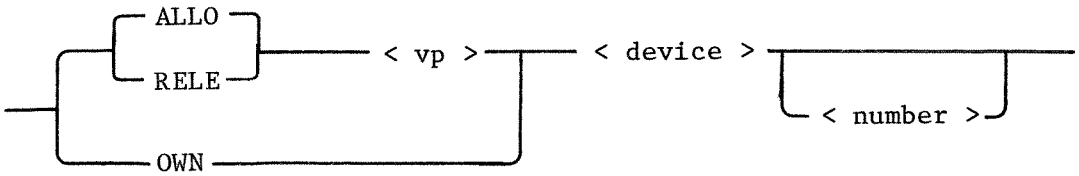
< register >



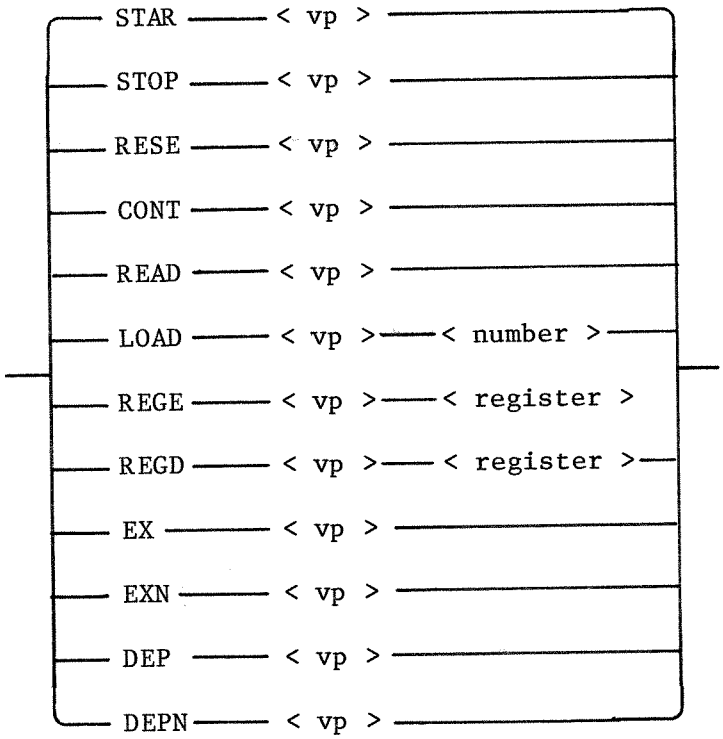
< device >



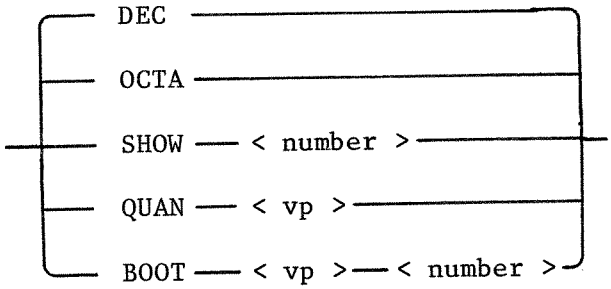
< device command >



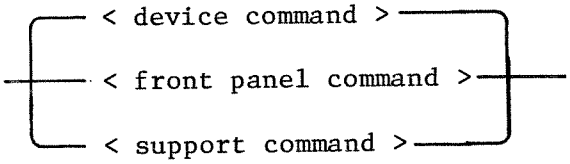
< front panel command >



< support command >



< vmm command >



BIBLIOGRAPHY

1. Bagley, J.D., Floto, E.R., Hsieh, S.C., and Watson, V. "Sharing Data and Services in a Virtual Machine System", Operating Systems Review, Vol. 9, No. 5 (1975), pp. 82-88.

2. Belpaire, G. and Hsu, N.-T. "Formal Properties of Recursive Virtual Machine Architecture", Operating Systems Review, Vol 9, No. 5 (1975), pp. 89-96.

3. Brinch Hansen, P. Operating System Principles. Prentice-Hall, Englewood Cliffs, New Jersey (1973).

4. Buzen, J.P. and Gagliard, U.O. "The Evolution of Virtual Machine Architecture", National Computer Conference Proceedings, AFIPS Press, Vol. 42 (June 4-9, 1973), New York.

5. Data General Corporation. "Programmer's Reference Manual: Nova Line Computers", No. 015-000023-03, Rev. 3, Westboro, Mass. (January, 1976).

6. Data General Corporation. "User's Manual: Introduction to Programming the Nova Computers", No. 093-000067-01, Rev. 1, Westboro, Mass. (September, 1972).

7. Data General Corporation. "User's Manual: Programmer's Reference: Peripherals." No. 015-000021-04, Rev. 4, Westboro, Mass. (January, 1977).

8. Goldberg, Robert P. "Survey of Virtual Machine Research", IEEE Computer, Vol. 7. No. 6 (June, 1974), pp. 34-45.

9. Goldberg, Robert P. "The Architecture of Virtual Machines", National Computer Conference Proceedings, AFIPS Press, Vol. 42 (June 4-9, 1973), New York, pp. 309-318.

10. Meyer, R.A. and Seawright, L.W. "A Virtual Machine Time-Sharing System", IBM Systems Journal, Vol 9, No. 3 (March, 1970), pp. 199-218.

11. Parmlee, R.P., Peterson, T.I., Tillman, C.C., and Hatfield, D.J. "Virtual Storage and Virtual Machine Concepts", IBM Systems Journal, Vol. 11, No. 2 (February, 1972), pp. 99-130.

12. Peterson, James L. "A Compiler for a PASCAL-Like Language." Systems Programming Laboratory Note 1, Department of Computer Sciences, University of Texas at Austin (August, 1977).

13. Peterson, James L. "Code Generation for a PASCAL Compiler for a Nova Computer", Systems Programming Laboratory Note 2, Department of Computer Sciences, University of Texas at Austin (Septemper, 1977).

14. Peterson, James L. "Using PASCAL on the Novas", Systems Programming Laboratory Note 4, Department of Computer Sciences, University of Texas at Austin (January, 1978).

15. Popek, Gerald J., and Goldberg, Robert P. "Formal Requirements for Virtualizable Third Generation Architectures", CACM, Vol. 17, No. 7 (July, 1974), pp. 412-421.

16. Popek, G.J. and Kline, C.S. "The PDP 11 Virtual Machine Architecture: A Case Study", Operating Systems Review, Vol. 9, No. 5 (1975), pp. 97-105.