SU-326-P.39-30

A NOVA 3 EMULATOR ON THE STANFORD EMMY SYSTEM

by

Daniel R. Hafeman

June 1978

TECHNICAL NOTE NO. 141

Digital Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305

The work described herein was supported in part by the Department of Energy under Contract No. EY-76-S-03-0326-PA 39.

Digital Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, CA 94305

Technical Note No. 141

June 1978

A NOVA 3 EMULATOR ON THE STANFORD EMMY SYSTEM

by
Daniel R. Hafeman

ABSTRACT

A NOVA 3 Emulator based on the EMMY host machine and the Stanford Emulation Laboratory is described. This is a "class A" code emulator (emulates user accessible aspects of the NOVA 3) and includes a basic set of emulated peripherals, such as paper tape, TTY and real-time clock. Of particular interest is the software interface between the emulator and the laboratory peripheral system via the UNIX operating system. Through this interface the emulator user may employ the full power of UNIX to create, organize and access file, while using the EMMY host for efficient emulation.

Instruction times for the NOVA emulation are provided together with an analysis of the host's effectiveness in supporting the emulation. An appendix describes the emulation of the NOVA 3 front panel for laboratory users interested in operating the emulator.

The work described herein was supported in part by the Department of Energy under Contract No. EY-76-S-03-0326-PA 39.

1.Ø INTRODUCTION

This paper describes the emulation of a Nova 3 computer on the Stanford Emmy System. The Nova architecture, when contrasted with other target machines currently running on Emmy, displays some unique architectural characteristics. It is not a register oriented machine (only four accumulators) and has a very simple state structure; specifically only three state flags. Because of this simple organization, condition flags like underflow, zero, etc., do not exist and therefore condition testing must be performed within the instructions that generate the conditions. Therein lie the problems that make "soft" Nova emulation extremely dificult and slow.

This discussion focuses on the implementation of the Nova 3 code emulator. Two accompanying papers describe the Cartridge Disk Interface and the Console Emulator.

2.0 NOVA 312 CODE EMULATOR

2.1 The Target Machine

Three versions of the emulated Nova (Novaem) have been implemented; the first two are paper tape systems, and the third contains a 4047A cartridge disk system. They are:

1) Novem - a Nova 312 using the Datapoint Genacc (1) program to simulate teletype, paper tape reader, and paper tape punch IO devices.

2) NovaemU - A Nova 312 using the recently implemented Mini UNIX (2) interface to simulate teletype, paper tape reader, and paper tape punch IO devices.

3) NovaemCD - a Nova 312 with a 4047A cartridge disk system installed. It uses the Mini UNIX system to simulate IO devices.

Each of the machines represents a fully implemented Nova 3 minicomputer with hardware multiply/divide options installed. All of the machines support:

a) 32K words of memory,

b) ASR33 teletype: NovaemU and NovaemCD support reader/punch. Novaem provides a teletype without reader/punch.

c) DG paper tape punch,

d) DG Paper tape reader,

e) Real time clock,

f) Hardware multiply/divide,

g) A complete Nova 3 console.

Novaem fully represents (class A) the Nova instruction set as described in the Data General Programmer's Reference Manual. No additional discussion is required here.

There are limitations to the emulation specifically in the I/O area. They are:

a) The Nova console is only a functional emulation of the Nova 3 console. See Console Emulator User's Guide.

b) The Real time clock provides timing intervals of 83, 500, 50 and 5 ms as compared to 16, 100, 10 and 1 ms for a real Nova. The times were increased to compensate for instruction execution speed differences.

c) I/O timing is significantly different since I/O devices are emulated by the Datapoint and UNIX systems.

d) The code emulator runs at approximately 10% of the speed of a real Nova 3.

Novaem supports neither the memory mapped unit (MMU), nor the power fail option. The power fail flag is wired off (power never fails).

3.0 IMPLEMENTATION

3.1 State Mapping

key to the design of any emulator is the representation of target state in the Host. This data structure greatly influences the performance of the emulator. Ideally all Nova registers would map into Emmy registers and all Nova state flags would be represented by equivalent Emmy flags.

An early design of Novaem attempted to use the ICODE field of Emmy RO to contain Nova state flags. This failed for two reasons:

1) Since Icode fields are swapped on subroutine calls, any procedure called by a procedure could not gain acess to the state information. Thus only a single level of subroutine nesting could be allowed.

2) Interrupt handlers have no access to the state for the same reasons.

This seems to be a severe limitation of the Emmy since there is no other fast mechanism available to perform state testing. A global ICODE space is needed.

Novaem attempts to create such a space by reserving R2 as the state register as shown in figure 3-1. Bits 16-23 of R2 contain state information tested regularly. Since these bits correspond to the Icode field of RO a "fast" insert may be performed prior to testing. For interrupt synchronization reasons, R2 is fixed and may not be temporarily swapped out by any routine.

R1, bits 14-0, contain the Nova PC. Bits 15-23 must = 0, and bits 24-31 define the Emmy main memory command field which specifies; 2 byte addressing, a two byte data word, right justified, and sign extended. All Emmy main memory accesses use this format.

R5 is reserved as the instruction register, IR. All instructions are parsed left to right as shown in figure 3-2. Decoding is implemented with the double register left logical shift instruction using R4 and R5 as the register pair.

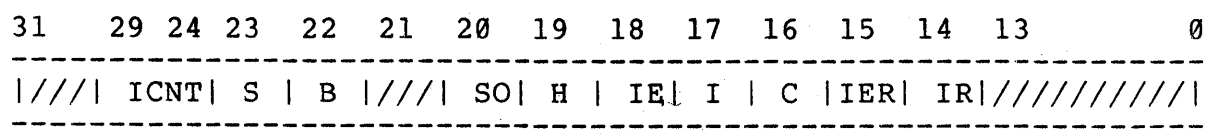
R3, R4, and R6 are general purpose registers.

R7 is reserved for future software probes.

The Nova accumulators, stack pointer, and frame pointer reside in locations 0-5 of Emmy control store. The 16 bit Nova registers are right justified. Bits 16-31 of each word remain unused and take on no assumed value.

Locations 7-16 Hex (CR0-CR15) provide scratch storage for storing Emmy state on subroutine calls. Any number of subroutine levels are allowed but all subroutines must cooperate in the use of the storage

STATE (EMMY R2):



**IR --Interrupt request; Emulates Nova I/O Bus Interrupt line. It is set by any I/O device that enters an interrupt request, i.e. increments ICNT.

IER --Interrupt enable request; Managed by INTEN/INTDIS instructions.

C --Carry.

**I --Interrupt FLAG [redacted]; I<- IR during instruction decode.

IE --Interrupt enable; IE <- IER during instruction decode.

H --Halt.

SO --Stack overflow flag; Managed by MØ3 stack instructions. When SO <- 1 then I,IR <- 1 and ICNT <- ICNT +1.

B --Busy; Managed by M63WAIT. When Busy=Ø, Novaem is in console emulator mode and is halted. When Busy=1, Novaem is either in reset or code emulation modes.

S --Start request; Set by MØØ, console emulator, when code emulation is to be resumed. It is cleared by M63WAIT prior to entering MØØ, code emulator.

**ICNT--A count of the number of interrupt requests pending. When ICNT >Ø, then IR=1, and when ICNT=Ø, then IR=Ø and I=Ø.

**Note: An I/O device sets IR and increments ICNT when issuing an an interrupt. I<- IR at the beginning of MØØ. To retract an interrupt request, a device must decrement ICNT and if ICNT=Ø, then both I and IR must be cleared. This register may be inspected while in console emulator mode by typing: HR2/ . The Hex contents will be displayed.

Figure 3-1 Novaem State Register

since no stack structure exists. This means that a routine must be aware of the memory and register resources used by any routine it calls.

Locations 17-25 Hex (IR1-IR15) are scratch storage for a single level of interrupt. Therefore, Emmy interrupts must be disabled during the whole of any interrupt procedure.

The I/O table starts at Hex 80 as illustrated in figure 3-5. Section 3.3 describes I/O mapping in detail. It is important here to note that the device I/O registers, RegA-RegC, reside in this table. The last entry in the table represents the CPU IO functions; namely the console switch register, address register, and display register. These are maintained by the console emulator and used by the CPU (device = 77) I/O instructions. The first entry into the table, a record of zeroes, is the I/O table for all non installed devices. It is currently used only as a header into the I/O table.

Figure 3-3 summarizes the above discussion by presenting an Emmy control store memory map. Notice that the code emulator starts at Hex 400.

3.1.2 Nova State Register

As stated earlier, Emmy R2 is used as the Nova State Register. All global CPU state information is found in this register. This state is now defined (see figure 3-1).

IE, IER (bits 18,15), interrupt enable and interrupt enable request. IE is the Nova interrupt enable bit. It is updated from IER prior to instruction decode. IER is managed by the CPU I/O instructions.

I, IR (bits 14,17), interrupt and interrupt request. I is the Nova interrupt flag. If interrupts are enabled (IE=1) and I=1 then the interrupt u-code is executed at competion of the current instruction. I is updated from IR prior to instruction decode. IR and ICNT describe the state of the interrupt request line on the Nova I/O Bus. When IR=1, one or more devices are requesting an interrupt and ICNT (bits 24-29) defines the number of requesting devices. If ICNT=0 then I, IR must = 0. If ICNT >0 then IR must = 1.

To issue an interrupt request, a device must increment ICNT and set IR=1. To clear the request, the device decrements ICNT and if it = 0 proceeds to clear IR and I.

C (bit 16), Nova carry flag. Carry is maintained by the Nova arithmetic and stack instructions.

SO (bit 20), Nova Stack Overflow flag. SO <- 1 when a stack overflow has been detected (see Nova Programmer's Reference Manual). When SO is set, during a stack instruction, both I and IR are also set and ICNT incremented. If interrupts are enabled, the stack overflow trap procedure is executed at completion of the current instruction.

S, B, H (bits 23, 22, 19); Start, Busy, and Halt. These bits define the running status of Novaem. If Novaem is not in console emulator mode, then Busy will = 1. Halt <- 1 when a request to enter console emulator mode has been received. This can occur in two ways; 1) a halt instruction is executed, or 2) the operator performs a console break function at which time the Datapoint or UNIX issues a halt u-interrupt to set H. If H=1 at the completion of the current isntruction, an exit is made to the console emulator and B <- Ø.

The console emulator starts a Nova Program running in one of 3 ways using start and halt:

1) S=1, H=1 (single step). Start is cleared at instruction fetch time. At completion of the current instruction, an exit to the console emulator is made since H=1.

2) S=1, H=Ø (continue). Start is cleared at instruction fetch time. Program execution will continue until H<-1.

3) S=Ø, H=Ø and procedure M62ST is executed (start). A Nova start function is performed as described in the Programmer's Reference Manual. Execution will continue until H<-1.

In all cases the Busy flag = 1 during execution.

3.1.3 Nova Macro States

Three Macro states are defined for Novaem:

1) Reset. Novaem is executing the initialization procedure; Busy=1.

2) Console emulator. Novaem is emulating the Nova Console and is not executing code; Halt=1 and Busy=Ø.

3) Running. Novem is performing code emulation; Halt=Ø and Busy=1.

3.2 Code Emulator

Figure 3-4 is a flow diagram of the Novaem code emulator and Table 3-1 lists all procedures used in Novaem.

3.2.1 Instruction Sequencing and Control.

Routines M6Ø-M64 perform all instruction sequencing and initial instruction decoding. This code was designed with speed as the key design goal since some or all of these routines must be executed once per Nova instruction.

M6Ø starts the instruction cycle by updating the interrupt and interrupt enable flags from their respective request flags (section 3.12). Next, the instruction, already in IR (Emmy R5), undergoes

initial decoding and a branch is made to the appropriate execution routine (see table 3-1). All instructions return to M60RET1 - M60RET4 to increment the PC and start the next instruction fetch. At this point, while the fetch is in progress, the halt and interrupt flags are examined. If neither is set, the program loops back to the beginning of M60. Otherwise, further testing is performed to determine which flag is set. Halt causes a branch to M63 which in turn clears Busy and exits to M80, the console emulator. If Halt =0 and Interrupt =1, M64 performs the interrupt sequence before returning to M60. This involves fetching the instruction at (0).

Two interesting observations should be noted:

1) The PC is incremented at completion of the instruction cycle since all relative address calculations use the address of the current instruction as the base.

2) The state control flags are sampled while the next instruction fetch is in progress. This allows normal interrupt free sequencing to overlap instruction fetch at the expense of an extra instruction fetch when an interrupt does occur, or a dummy fetch if halt is set.

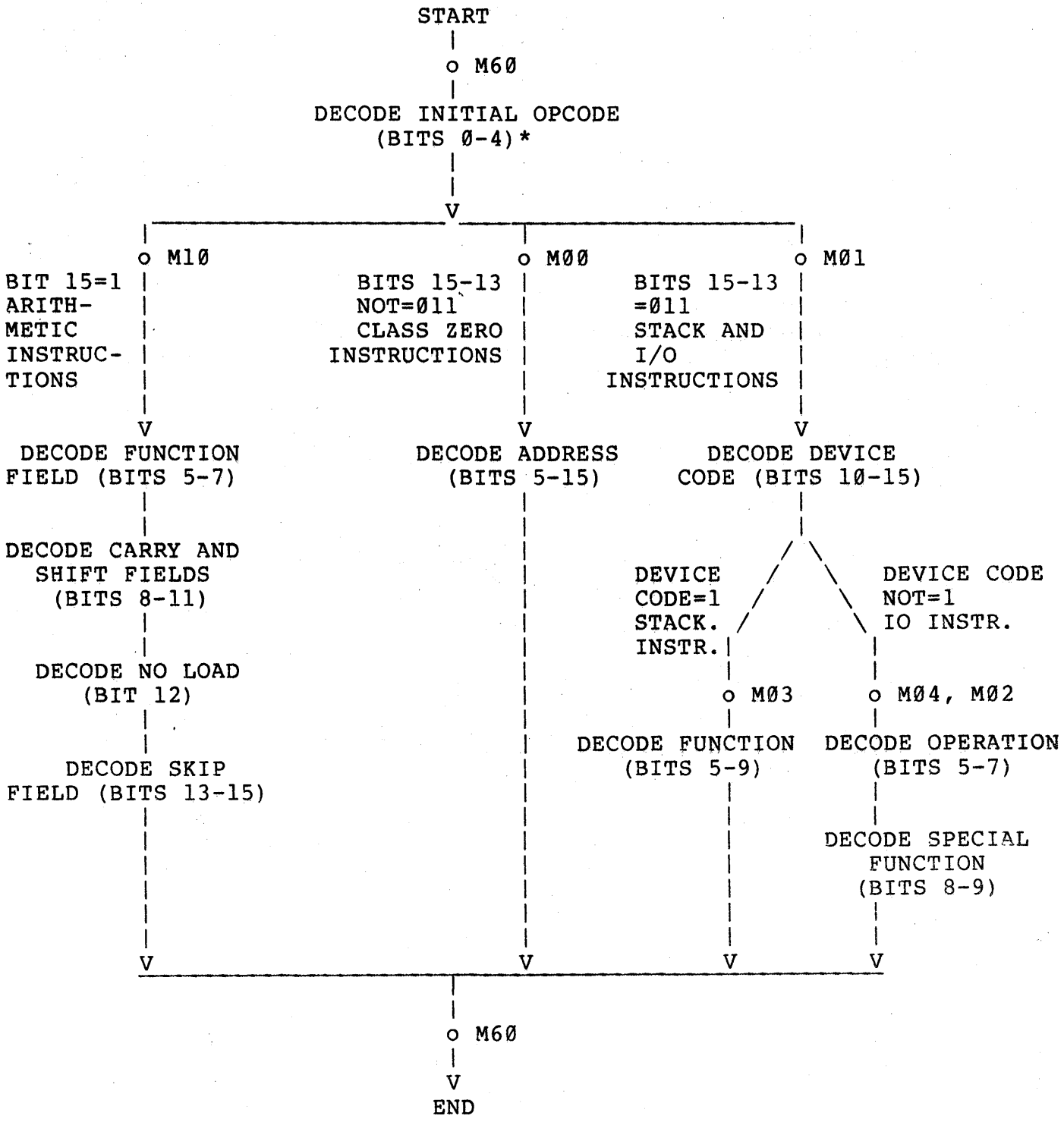
Since five Emmy instructions are executed between starting the fetch and examining IR, Emmy main memory access speed is not a performance consideration here.

3.2.2 Instruction Execution

The execution units defined in table 3-1 perform further decoding and execution of the instructions. Except for the I/O execution routine (M04), speed was again the principal design consideration. However, here is where speed degregation is at its worst.

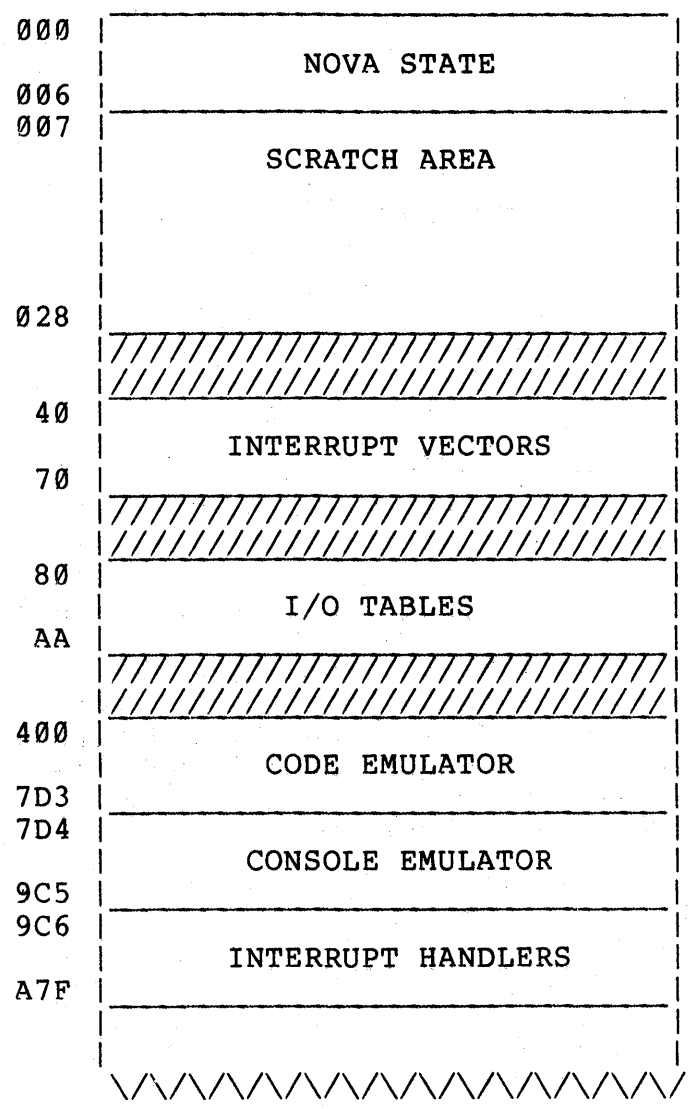
M10, arithmetic instructions, must decode four additional fields to complete the instruction (figure 3-2). One can combine fields at the expense of redundant code resulting in fewer decode steps. This was done, for example, with the carry and shift fields in M10. Any further combination would have resulted in an exponential growth in code redundancy. For example, if the skip field is combined with the carry preset and shift options, a 7 bit field results with 128 almost identical execution routines required. M10, alone, would require 1/2 K of storage. However, only two Emmy Instruction times (1.4 us) can be saved.

M00, M03, and M10 avoid the use of subroutines at the expense of code redundancy. M10, however, does use the address generation routine, P00, to calculate absolute Nova addresses; the code redundancy would have been excessive otherwise. P00 must perform bounds checking on each address to facilitate the auto increment/decrement features of the Nova memory; a function that clearly needs hardware assistance. Three Emmy instructions (2.1 us) are required just to determine that the address is not in the auto increment/decrement range. If it is in range, an additional 4 instructions may be required to determine if an increment or decrement is to be performed.



*Bit numbering uses DG standard; Bit 0 = left most bit.

Figure 3-2. Instruction Decoding Chart



Emmy Registers:

R1 - Nova PC
R2 - Nova State
R5 - Instruction Reg
R3, R4, R6 - Scatch Registers

Figure 3-3. Memory Map for NovaemU

MAIN ROUTINES

NAME	DESCRIPTION
MØØ	Executes class zero non I/O instructins (Bit 15 of instruction=Ø).
MØ1	Decodes all I/O, CPU, and stack instructions.
MØ2	Executes CPU instructions.
MØ3	Executes all stack instructions.
MØ4	Executes all non CPU instructions: MØ4TTYI - teletype keyboard MØ4TTYØ - teletype printer MØ4PTR - paper tape reader MØ4PTP - paper tape punch MØ4RTC - real time clock MØ4CDS - cartridge disk system.
MØ5	Executes Nova hardware multiply.
MØ6	Executes Nova hardware divide.
M1Ø	Executes class 1 instructions (Bit 15 of instr. = Ø).
M6Ø	Performs initial instruction decoding, instruction sequencing, and control.
M62	Novaem initialization routine: M62ST - emulates Nova start function M62RS - emulates Nova reset.
M63	Provides the code emulator interface to to the console emulator.
M8Ø-M4ØØ	Console emulator.
IØØ-I127	U-interrupt handlers.

Table 3-1. List of Nova Routines and Procedures

KEY PROCEDURES USED IN NOVAEM

NAME	DESCRIPTION
P00,P03	Forms absolute Nova main memory address calculation.
P01	Executes Nova I/O clear function for simple device interfaces.
P02	Sets done and issues Nova interrupt for simple devices.
P04	Emulates the Nova IORST pulse on the I/O bus.
P05-P13	Console emulator procedures.
P14-P15	Saves/restores Emmy registers in/from IR0-IR15.
P16WFF	Used in NovaemU and NovaemCD to wait for Emmy-to-UNIX mailbox to become available.
P21-P25	Cartridge disk emulator procedures.

Table 3-1. List of Nova Routines and Procedures - Continued.

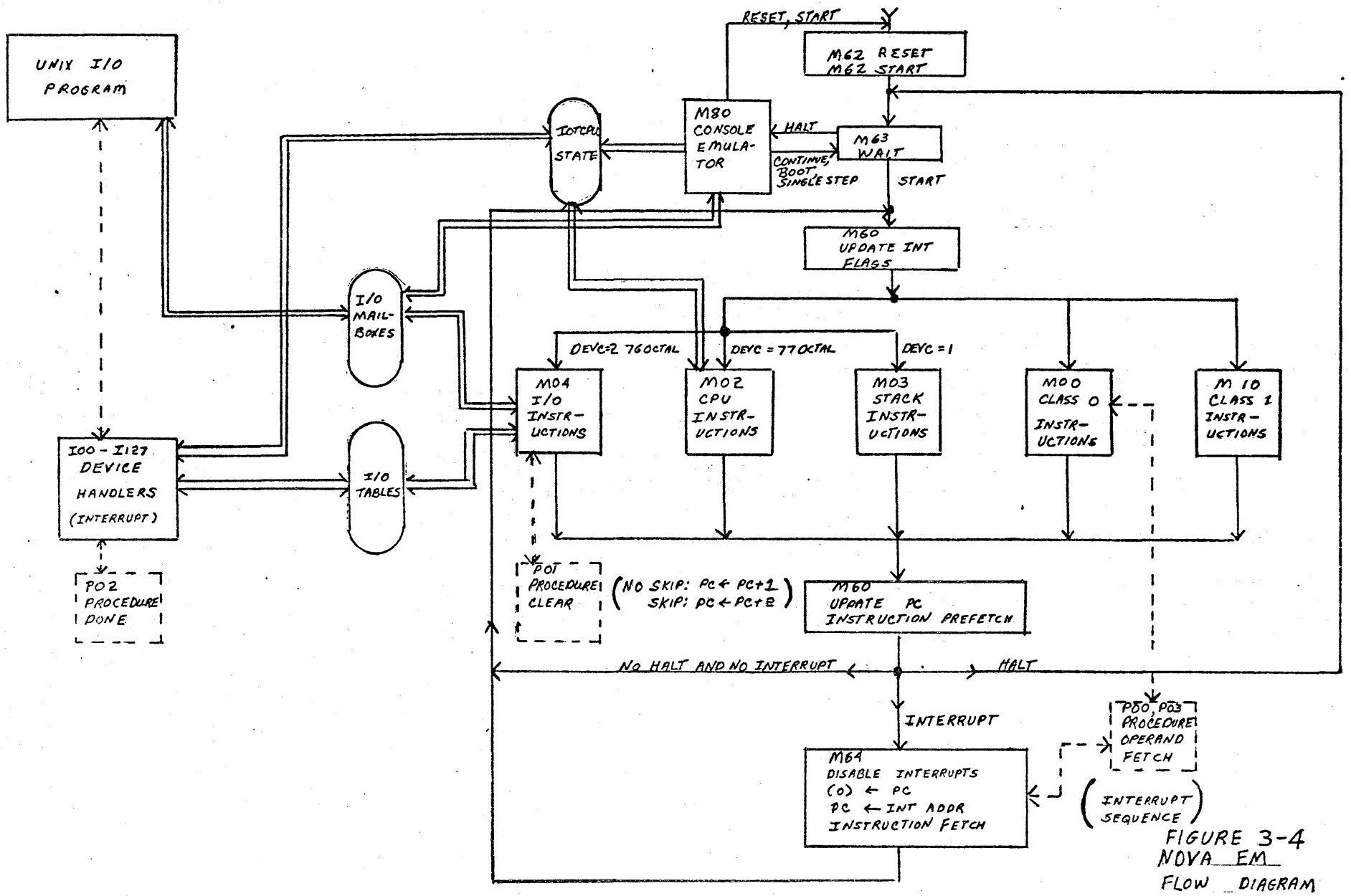


FIGURE 3-4
NOVA EM
FLOW DIAGRAM

The Stack instructions also suffer performance degregation, due to decoding, since this class can only be detected by observing that bits 0-4 define an I/O instruction and that bits 10-15 specify device code 1. Then an additional decode step is required to specify the instruction within the class.

The I/O instructions, supposedly executed relatively infrequently, attempt to save space by the use of subroutines wherever possible. These instructions interact with the u-interrupt handlers via the I/O table, discussed in the next section.

3.3 I/O Structure

The most complex data structure in Novaem is the I/O table; it essentially emulates the Nova I/O Bus. Figure 3-5 shows the organization of this table.

Each I/O record represents a single device on the Nova Bus, and the order of the records in the table determines the relative priority of the I/O devices. The Record consists of six Emmy words.

The first 3 words map one to one with the device I/O registers, RegA-RegC, and are used by the Nova DIA-DOC instructions. Bits 16-31 of these words are generally not used.

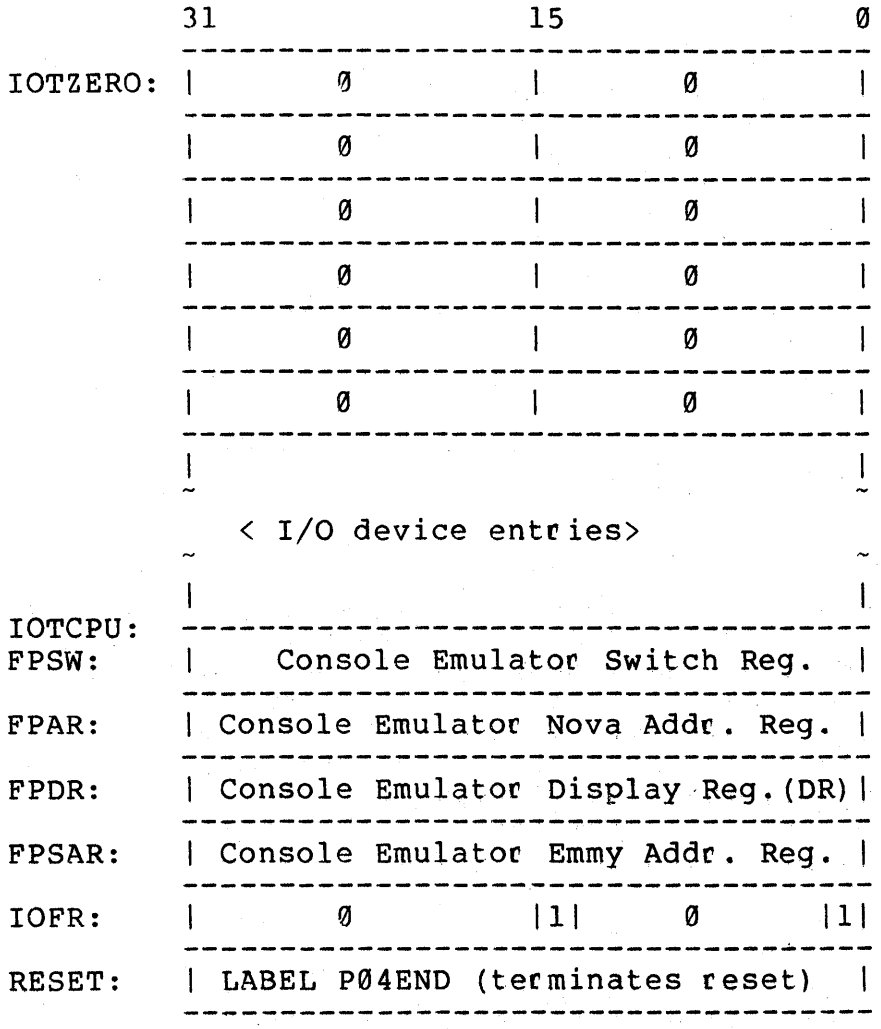
The fourth word, called Reg Temp, provides temporary status information and is used only by the CPU and Cartridge Disk I/O emulators.

The fifth word (IOFR) contains the state of the I/O device as seen by the Nova I/O Bus. The device busy and done flags correspond to the same flags in the real Nova. Bits 8-13 define the I/O bus device code.

Bit 0, interrupt, is set whenever an interrupt request has been issued by this device; that is, the device u-interrupt handler has incremented ICNT in R2. Bits 16-31 define the done field (DONEF). When the done flag is set, a bit specified by the device's I/O mask bit assignment (see Nova Peripherals Manual), is also set in this field.

For example, the paper tape reader is assigned mask bit 11 in the DG peripheral structure. Therefore, when a read operation completes, the paper tape reader u-interrupt handler sets both the done bit and bit 20 in DONEF. The MSKO instruction uses DONEF to determine which new devices will be generating interrupt requests and which old devices must retract their requests when the I/O mask is changed.

The final word of the set contains the address of the initialization routine that is to be executed for this device during I/O reset (IORST). Procedure P04IORST scans through the I/O table executing each device's reset routine. Most peripherals use procedure P04CL as their reset procedure and thus the last word in their I/O records contains the label P04CL.

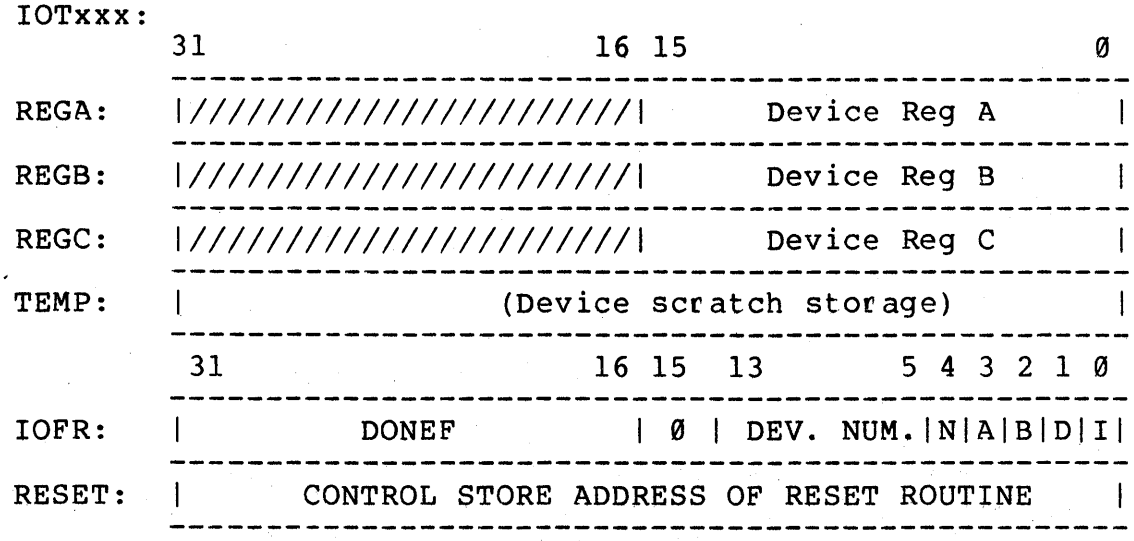


IOTZERO --Serves as the table header record and contains all zeroes.

IOTCPU --Serves as the termination record of the table and as the console emulator scratch area. Bit 15 of IOFR must = 1. It terminates the MSKØ execution routine. Bit Ø of IOFR always equals 1 (interrupt always pending) to terminate the INTA interrupt search loop. RESET contains the label, PØ4END, which terminates the IORST reset loop.

Figure 3-5 IOT Table (Page 1 of 2)

IOTxxx:



xxx --Name of device. For example, IOTPTR is the paper tape reader entry.

I --Interrupt issued; I=1 when the device has incremented ICNT and set IR.

D --Nova I/O Bus Done flag.

B --Nova I/O Bus Busy flag.

A --Abort; A=1 when an abort is in progress on this device. It is cleared when abort finish mail arrives.

N --Not UNIX device. N=1 when the device does not use UNIX services. The real time clock is such a device.

DONEF --Done flag field. When done is set, a bit corresponding to the device's mask bit is also set in this field.

DEV NUM--Contains the UNIX device number for this peripheral if N=Ø. Bits 8-13 define the Nova I/O device code. Thus, each peripheral has a three octal character UNIX device #. The two most significant characters define the Nova device code. Up to 8 UNIX devices, then, may be assigned to the same Nova device code.

NOTE: Bit 15 of IOFR must = Ø for all entries in IOT except for IOTCPU where this bit = 1.

Figure 3-5 I/O TABLE (Page 2 of 2)

The I/O mask word facilitates the emulation of the Nova software controlled priority interrupt system. As stated earlier, it is maintained by the MSKO instruction execution unit. A device u-interrupt handler issues a Nova interrupt (increments ICNT and sets IR in R2) whenever its DONEF field "anded" with IOMSK yields a non zero result.

3.3.1 Device U-Interrupt Handlers

Routines I00-I127 form the set of device interrupt handlers. Unlike the PDP-11 Emulator (2), all interrupt processing is performed by the appropriate u-interrupt handler when the interrupt is received. Since only a single layer of interrupt nesting is allowed, Emmy interrupts must remain off during this process.

The u-interrupt handlers must update device registers as required and update the devices I/O status. This generally involves setting the done flag, clearing the busy flag, and issuing a Nova interrupt if the device's interupt logic is enabled; that is, if the entry in DONEF shines through the current mask word.

At completion of interrupt processing, the UNIX or Datapoint mailbox busy flag is cleared and the previous Emmy process restored.

4.Ø EVALUATION OF NOVAEM

4.Ø Validation

Validation of Novaem constituted an extremely important phase in the development of Novaem. Since the only specification of the machine available is the Programmer's Reference Manual, there is no precise description of the Nova 3 to which Novaem can be compared.

Validation was performed in 3 steps:

1) Sample code sequences defined in the Programmer's Reference Manual were coded and executed, and the results carefully compared with those given.

2) The Nova 3 functional diagnostics tests were successfully run. These included the TTY functional diagnostics, Nova Instruction Exerciser, and the Nova Arithmetic Tests. These programs perform extensive instruction, addressing, and bounds checking.

Probably the extra time required to design a class A code emulator is more then compensated for during validation. The author has found that testing an emulator without the aid of diagnostics is at least as challenging and time consuming as the design and implementation of the emulator.

3) Actual user programs were run. Specifically Single User Basic and the SOS Editor were tested.

4.1 Performance

Instruction execution speeds were measured by an evaluation program which executed each instruction, in a loop, a fixed number of times. The Nova real time clock was used as the time base. Table 4-1 shows the results of this test and, from the table, one can conclude that Novaem runs at 8 to 1Ø% the speed of a real Nova.

Notice that Novaem compares most favorably with the Nova 3 on instructions that have low decoding overhead and high function content. For example, Novaem executes the RET instruction only 5.4 times slower then the Nova, but executes the primitive instruction, ADD, 19 times slower. This is additional evidence that the Emmy is more efficent at instruction execution then at instruction decoding.

The average Emmy instruction rate was measured directly using a counter with Novaem executing the diagnostics and Single User Basic. Dividing each Nova instruction execution time by the average Emmy instruction execution time gives the average number of Emmy instructions executed per Nova instruction.

4.2 Program Size and Efficency

INSTRUCTION	EMULATED TIME IN US	NOVA 3 TIME IN US	AVE # OF EMMY ** INSTR/NOVA INSTR
LDA,STA	15.7	2.0	21.7
ISZ	18	2.4	24.8
JMP	14.55	1.0	20.1
JSR	16.65	1.2	23.0
For each level of indirection add	3.5	1.0	4.8
For auto increment and decrement add	8.8	1.4	12.1
For Skip add	1.0	.2	1.3
COM, MOV	18.4 to 20.7	1.0 to 1.3	28.5 to 30.0
NEG, INC, ADD	19.5 to 21.8	1.0 to 1. 3	26.9 to 30.1
AND	19.9 to 22.0	1.0 to 1.3	27.5 to 30.4
ADC, SUB	20.9 to 22.6	1.0 to 1.3	28.9 to 31.2
Variations are due to options			
MUL	27.5	5.8	38.0
DIV	29.4	5.8	40.6
PSHA	16.9	1.9	23.3
POPA	15.7	2.1	21.7
SAV	35.2	6.5	48.3
RET	33.6	6.5	46.4
MTFP, MTSP	13.4	1.0	18.5
MFFP, MFSP	13.8	1.0	19.1
DIA	14.9	2.2	20.6
DOA	23.0	2.2	31.77
MSKO	46.2	2.2	64.0
INTA	23.2	2.2	32.2
INTEN/INTDIS	13.8	2.2	19.1
READS	14.6	2.2	20.2

Table 4.1 Novaem Performance (PAGE 1 OF 2)

** Based upon Novaem measurements, the average Emmy instruction period = 724ns.

Average Measured Execution Performance:

PROGRAM	NOVA INSTR. RATE	AVE # OF EMMY INSTR PER NOVA INSTR	AVE EMMY INSTR. PERIOD
Code Exercisor	51 KIPS	27.8	714 ns
Arithmetic Test	49 KIPS	28.5	716 ns
Single User Basic	52.5 KIPS	26	735 ns

Table 4.1 Novaem Performance - Continued

The Nova Code Emulator, not including the Cartridge Disk system, requires 980 words of program storage along with another 40 words of data storage. The entire program including u-interrupt handlers and the Console Emulator, uses 1663 words of storage. Notice, in figure 3-3, that the Console Emulator requires almost as much storage as the entire code emulator.

The following Emmy code efficiency numbers were determined by randomly sampling the static code in the Nova Code Emulator:

A-occupancy - 98%
T-occupancy - 82%
Full usage - 80%
Total bit occupancy = 91%.

These results indicate that the storage efficiency of Novaem isn't too bad and considerably better then that reported for the PDP-11 emulator. The reader should beware, however, that these numbers can be deceiving. The author, while coding Novaem, often made use of otherwise unused T instructions to preset registers for possible later use--often not needed. However, such instructions were reported as having full usage.

5.0 IMPROVEMENTS

5.1 Emulator Optimization

Novaem's performance is acceptable in the research environment for which it was designed. However, as a Nova 3 computer, its execution speed is clearly unacceptable. The redesigned Emmy alleviates the speed problem somewhat by offering a factor of 3 speed improvement. Careful recoding of Novaem could yield an additional 10% improvement. However, even with both improvements, Novaem still doesn't match the performance of a real Nova 3. It is clear from previous discussion that Novaem needs hardware assistance in performing instruction decoding. Without it, one can avoid multiple decoding steps only by processing the entire instruction as one opcode field, requiring a sparse 64K word jump table. A personality module is required to perform front end instruction processing. This module, for example, would detect the stack instructions by simultaneously monitoring the device code and opcode fields of the class zero instructions. It would present a single well defined high entropy opcode field to Emmy. In additon, this module would perform the carry preset and shift functions of class A instructions.

5.2 Emmy Optimization

Several ideas for improving Emmy performance have been collected over the course of the project. Some of these are listed below:

1) Emmy needs a bit addressable register for containing target state information, that isn't disturbed by subroutine calls/returns or interrupts. The register should be available as an operand to the condition testing instructions, like the ICODE and CCODE fields of R0.

2) Some mechanism is required to specify the size, in bits, of the registers to be used in arithmetic operations that set flags; for example, a carry out of the 16th bit should set the Emmy carry flag in a 16 bit addition. Novaem spends considerable time extending 16 bit operands to 32 bits so that the Emmy condition codes can be used.

3) Novaem must mask out bits 16-23 of its program counter on each increment in order to emulate the 15 bit Nova program counter. This step alone is a .75 us operation. Dedicating a register, in hardware, as the target program counter, and specifying its size in software would eliminate this step.

4) Of course the obvious things such as A and T machine parallel operation, a wider control word (64 bits), and more Emmy registers would surely improve performance.

6.Ø CONCLUSIONS

The project was an incredible learning experience. It pointed out, without doubt, that real time "soft" emulation of an existing architecture is a tough job. However, with a fast host and hardware assistance in decoding, it still may be feasible. A writable PLA chip would be really useful. With it, one could exploit the benefits of a hardware personality card, but still have a totally soft machine since the emulator u-code would program the PLA at initialization.

REFERENCES

1) GENACC Program Listing
Digital Systems Lab, Stanford Electronics Lab
Stanford University, March 1978

2) Charles Neuhauser
"An Emmy Based PDP11/20 Emulation"
Digital Systems Lab, Stanford Electronics Lab
Stanford University, March 1977

3) Charles Neuhauser
"Emmy System Processor -- Principles of Operation"
Digital Systems Lab, Stanford Electronics Lab
Stanford University, May 1977

4) Lee W. Hoevel
"Deltran: Principles of Operation. A directly Executed Language For Fortran II"
Digital Systems Lab, Dept. of Electrical Engineering and Computer Science
Stanford University, March 1977

5) Digital Equipment Corp
"PDP11/03 Processor Handbook"
Copyright 1975

6) Data General Corporation
"Programmer's Reference Manual--Nova Line Computers"
Doc # 015-000023-04
Copyright August 1976

7) Data General Corporation
"Programmer's Reference Manual--Peripherals"
Doc # 015-000021-00 Rev. 00
Copyright November 1974

8) Data General Corporation
"Exerciser for NOVA 3"
Doc # 096-000363-02
Copyright 1976

APPENDIX

NOVA 3 CONSOLE EMULATOR

1.Ø INTRODUCTION

Since the Nova Console Emulator is not a class A representation of the real Nova 3 front panel, the description of the console in the Programmer's Reference Manual should be discarded. The Nova console emulator uses the system teletype (or CRT) as the physical interface to the operator. The program is highly interactive, providing an interface similar to LSI-11 ODT (5), and as such may be easily learned.

This document provides a functional description of the console emulator.

2.Ø ENTRY INTO CONSOLE EMULATOR MODE

Novaem enters Console Emulator Mode when:

1) Initialization is completed following startup of Novaem.

2) A Nova halt instruction is executed by the code emulator.

3) The user issues a "break" command. The break command is performed by typing a Control X on UNIX or a "cancel" on the Data Point. The Halt flag is set in the Novaem State Register resulting in an exit to Console Emulator following completion of the current instruction.

4) The user issues a "hard reset" command. The Reset command is performed by typing a Control Z on UNIX or a Genacc Reset Command on the Data Point. An Emmy u-interrupt is issued causing Novaem to abort the current instruction and immediately execute the initialization code at which time an exit to the console Emulator occurs.

At entry to the console Emulator, Novaem will type the following message:

```
PC= aaaaaa DR= dddddd
@
```

"a" and "d" are octal digits. PC is the Nova Program Counter, and DR is the Nova Consle Display Register.

Note: The contents of DR are only important following a Nova Halt instruction. The Halt instruction AC field allows the programmer to specify an accumulator to be displayed at the console. Novaem accomplishes this function by setting DR = <selected ac> and displaying it at entry to the console emulator. The @ character is the console prompt indicating that Novaem is ready for user console commands.

While Novaem is busy, the console serves as the teletype device. Specifically, unlike a real Nova, the switch register cannot be modified. Only the Reset and Break commands are enabled.

3.0 FUNCTIONS

3.1 "/" Slash.

This command opens the memory location defined by the address descriptor immediately proceeding it. If no descriptor is provided then the last accessed Nova memory location is opened.

Four types of address descriptors exist.

1) Nova Memory

1-6 octal characters represents a 15 bit Nova memory address. The console will respond with the octal contents of the location.

2) Nova Internal Registers

ACO-AC3 - accumulators 0-3
SP - stack pointer
FP - frame pointer
SR - console Switch register (used by the READS instruction)
PC - Program Counter
FL - Carry, Stack OVF, and Interrupt enable flags

The console will respond with the octal contents of the selected register in all cases except for FL. If FL is selected, Novaem responds as follows:

@ FL/ C=<one binary digit> O=<binary digit> I=<binary digit>.

Where "C" is the carry flag, "O" the stack overflow flag, and "I" the interrupt enable flag.

3) Emmy Control Store

A CS<3 hex characters> addresses a control store location in the Emmy. The console responds with the hex contents of the location.

4) Emmy Registers

A HR<0-7> addresses an internal Emmy register. The console responds with the Hex contents of the location. This is especially useful in examining the State Register, HR2.

Example:

@ 157/ 177776

The user types 157 followed by the "/". The console responds with its contents.

If a "/" is typed with a null address descriptor, the console responds by printing the address of the last Nova memory location opened, echoes the "/", and prints its contents.

If an illegal address descriptor is entered, the "/" is ignored and the bell rung. The user should correct the error, or abort the line with a control Y .

3.2 "CR" Carriage Return

The carriage return closes the currently opened file, if no errors have occurred, and processes any command that may have been given.

Commands are:

1) Update contents of opened location.

The user opens a location as described above and then enters an octal or hex number, depending on the type of location opened, and types CR. The console responds with a CR LF and a new prompt. The contents of the opened location are updated and closed.

2) Update the Switch Register, SR, and enter the Nova Code Emulator: <Ø-6 character octal number><command>.

Valid commands are:

SS - Single step the Nova Processor. Novaem will exit console mode, execute the instruction pointed to by PC, and re-enter console mode as described in section 2.

ST - Start the Nova Processor. Novaem will perform a bus wide IORST, set PC=SR, and enter instruction Emulation mode.

CT - Resume execution. Novaem will enter instruction emulation mode fetching the instruction pointed to by PC.

BT - Perform Bootstrap. Novaem will perform the program load function described in the Programmer's Reference Manual.

RS - Reset the Code and I/O Emulators. Novaem will perform a bus wide IORST and return to console mode as described in section 2 above.

If the octal number proceeding the command is null then SR remains modified.

If an error is detected in the line typed by the user, the CR is ignored and the bell rung. The user should respond by either correcting the error or explicitly aborting the command with a control Y. Only 8 characters are accepted per entry. Any additional characters will be ignored and the bell rung. The console emulator

assumes that input streams are left justified. Thus, if an octal location is opened and an 8 character number is entered, then only the leftmost six characters will be accepted.

Example:

```
@ ACO/ 012345 1000ST CR
```

Upon detecting the CR, the console sets SR=1000 and exits to M62ST to perform the Nova start sequence which sets PC=SR. Thus the command effectively starts Novaem running at location 1000.

Example:

```
@ PC/ 000500 10SS CR
PC= 000501 DR= 000500
@
```

The single step command causes a single Nova instruction to the be executed and sets SR=10.

Example:

```
@ 157/ 177776 123 CR
@
```

If 157 is examined it will now contain 123:

```
@ 157/ 123
```

Example:

```
@FL/ C=1 O=0 I=1 010 CR
@
```

The carry flag will now =0, overflow will =1, and Interrupt Enable will =0. If less then 3 characters are typed before the carriage return, an error is indicated with a bell.

Example:

```
@100/ 012345 CR
@
```

The contents of 100 remain undisturbed.

3.3 "I" Close and Increment

Closes the currently opened location and displays the contents of the Nova memory location following the one last opened. The command functions like carriage return except that it only accepts the "Update Opened Location" command. All Code Emulator commands are rejected as errors.

Example:

```
@150/ 000001 123 I
151/ 125000
```

Location 150 will now = 123 and location 151 is opened. Notice that increment only operates on Nova memory addresses. For example, suppose AC0 is examined, after closing location 151, and closed with the I command. Location 152 will be opened next instead of AC1:

```
@AC0/ 177777 I
152/ 127000
```

3.4 "U" Close and Decrement

Like the "I" command except that the Nova memory location preceding that last opened is examined.

Example:

```
@150/ 000001 123 D
147/ 001000
```

"Control Y" Abort Current Line

The currently opened location is closed, unmodified, and the current line aborted. The console responds with *** CR LF @.

Example:

```
@123/ 177770 control Y ***
@
```

4.0 SWITCH REGISTER

While Novaem is executing code, the switch register is unavailable to the user. To update the SR while a program is in execution, the user should:

1) Enter a break (Control X).

2) Update SR

3) Resume execution with at continue, CT, command.

Example:

```
@Control X
PC= 001011 SR= 000000
@SR/ 000000 <desired contents> CT CR
```

Code execution will resume.

No internal state information is lost since the exit to the console emulator occurs at the completion of the current instruction.

REFERENCES

1) GENACC Program Listing
Digital Systems Lab, Stanford Electronics Lab
Stanford University, March 1978

2) Charles Neuhauser
"An Emmy Based PDP11/20 Emulation"
Digital Systems Lab, Stanford Electronics Lab
Stanford University, March 1977

3) Charles Neuhauser
"Emmy System Processor -- Principles of Operation"
Digital Systems Lab, Stanford Electronics Lab
Stanford University, May 1977

4) Lee W. Hoevel
"Deltran: Principles of Operation. A directly Executed Language For Fortran II"
Digital Systems Lab, Dept. of Electrical Engineering and Computer Science
Stanford University, March 1977

5) Digital Equipment Corp
"PDP11/03 Processor Handbook"
Copyright 1975

6) Data General Corporation
"Programmer's Reference Manual--Nova Line Computers"
Doc # 015-000023-04
Copyright August 1976

7) Data General Corporation
"Programmer's Reference Manual--Peripherals"
Doc # 015-000021-00 Rev. 00
Copyright November 1974

8) Data General Corporation
"Exerciser for NOVA 3"
Doc # 096-000363-02
Copyright 1976