ILLEGIBLE DOCUMENT

THE FOLLOWING DOCUMENT(S) IS OF POOR LEGIBILITY IN THE ORIGINAL

THIS IS THE BEST COPY AVAILABLE

A SOFTWARE SIMULATOR TO HOST THE DATA GENERAL NOVA ON THE IBM 360

by

SUNEE GADETRAGOON

B.Sc.(Hons.), Chulalongkorn University, Bangkok
1964

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1972

Approved by:

Major Professor

TABLE OF CONTENTS

INTRODUCTION

A simulator allows programs written for one computer to be run on another. So does an emulator. Because of this there has been some confusion about the difference. As a matter of fact some people use the terms interchangeably. Compounding this confusion, simulation also refers to an entirely different concept, mathematical modelling.[1]

The difference between the two is primarily the difference between the hardware and the software. An emulator can be implemented in two ways, by building an external hardware interface between the program input device and the computer or by changing the computer itself through microprogramming.

A simulator is a program written for the purpose of making the existing software and hardware compatible. Its advantage lies in the fact that no additional hardware needs to be built. So the computer requires no change that would alter its basic architecture, providing an additional measure of flexibility.

Software simulation of one computer on another can result in great gain with little effort. While the pitfalls of simulation are many, it is still the closest thing to a legal 'something-for-nothing' available to the computer user. It offers an opportunity of getting the available powers of one machine on another.

The purpose of this project was to provide a simulator which could be used to host the Data General Nova on an IBM 360. As described in the Systems Reference Manual For The Nova (2), the Nova computers are general purpose computer systems with a 16 bit word length. The machine is organized around four accumulators, two of which can be used as index registers. This accumulator/index register organization provides great efficiency and ease in

programming. The central processor performs a program by executing instructions retrieved from consecutive memory locations as counted by the 15 bit program counter PC. At the end of each instruction PC is incremented by 1 so that the next instruction is normally taken from the next consecutive location. Sequential program flow is altered by changing the contents of PC, either by incrementing it an extra time in a test skip instruction or by replacing its contents with the value specified by a Jump instruction. The other internal registers of importance to the programmer are four 16 bit accumulators, AC0 to AC3. Data can be moved in either direction between any memory location and any accumulator. Any instruction that references memory may address AC2 or AC3 as an index register. Instructions that move data to and from memory or the peripherals address a single accumulator as a source or destination of data while addressing a memory location or an I/O device. But the arithmetic and logical instructions do not reference memory; they simply address two accumulators, either or both of which may supply operands, and one of which may receive the result. Since an arithmetic or logical instruction does not contain a memory address, there are many bits that can be used for functions other than specifying the basic operation and the operands: the same instruction that adds or subtracts can also shift the result or swap its halves, test the result and/or carry for skip, and specify whether or not the result shall actually be retained.

Each instruction that addresses a memory location contains information to determine the effective address, which is the actual address used to fetch or store the operand or alter the program flow. The instruction specifies an 8-bit displacement which can directly address any location in four groups of 256 locations each. The displacement can be an absolute address, i.e. it may

be used simply to address a location in page zero, the first 256 locations in the memory. But it can also be taken as a signed number that is used to compute an absolute address by adding it to a 15-bit base address supplied by an index register. The instruction can select AC2 or AC3 as the index register; either of these accumulators can thus be used as an ordinary index register to vary the address computed from a constant displacement, or as a base register for a set of different displacements. The program can also select PC as a index register, so any instruction can address 256 words in its own vicinity (relative addressing). The computed absolute (15 bit) address can be the effective address. However, the instruction can use it as an indirect address, i.e. it can specify a location to be used to retrieve another address. Bits 1-15 of the word read from an indirectly addressed location can be the effective address or they can be another indirect address.

The program can make use of an automatic indexing feature by indirectly addressing any memory location from 20_8 to 37_8. Whenever one of these locations is specified by an indirect address, the processor retrieves its contents, increments or decrements the word retrieved, writes the altered word back into the memory, and uses the altered word as the new address, direct or indirect. If the word is taken from locations 20_8 to 27_8, it is incremented by 1; if taken from locations 30_8 to 37_8, it is decremented by 1.

In the early stages of this project it was attempted to translate the Nova Object Codes (NOC) into IBM 360 Assembly Language, but it was found that simulation in this way will put some unnecessary restrictions on Nova Assembly Language programmers. For example, in such a case it would be virtually impossible to code Nova Assembly Language instructions to modify other instructions of the same program.

These restrictions provided an impetus to design some software to simulate the Nova Central Processor, instead of the NCC, on the 360.

This report describes the operation and scope of a software package which effectively simulates the operation of the Nova Central Processor on the IBM 360, except for I/O operations. The software package consists of a single PL/1 program, which, when invoked, behaves as a Nova Central Processor and effectively simulates the execution of the NCC.

As Nova hardware processor is being simulated on the 360 software, it is not unexpected that the execution time of the NCC is considerably increased.

DESIGN OF THE SIMULATOR

The simulator basically consists of a variable size software-simulated Nova core and a software processor. The simulator reads in the core size in words -- each Nova word is 16 bits long -- and allocates appropriate 360 storage in units of 16 bits. Next it reads in the address of the memory starting from which the program is to be loaded, and the memory address at which the execution of the program is to begin. After this initialization phase the object code of the Nova program is read into the simulated memory and the execution is started as soon as the whole object module has been loaded. From the specified location it fetches one 16 bit instruction and interprets it in terms of the desired Nova operation and available System 360 facilities. The desired operation is then performed using 360 PL/1 instructions. The next instruction is then fetched, normally from the next consecutive software-simulated Nova word unless the immediately preceding operation itself dictates the location of the word from which the execution is to be resumed.

The simulation is terminated whenever any of the following conditions is met:

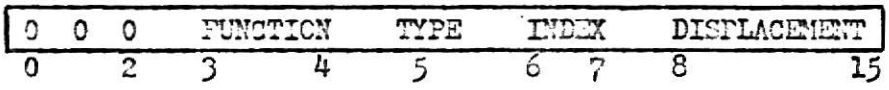
1. A HALT instruction is encountered.
2. An unidentifiable instruction is encountered.
3. The size of the object module exceeds the specified core-size.

At the termination of simulation the amount of core used by the object module, all the accumulators, the SHIFTER and the CARRY are printed out. If the termination occurs due to one of the first two conditions, the current value of the program counter is also printed out to show the last executed instruction.

NOVA OBJECT CODE

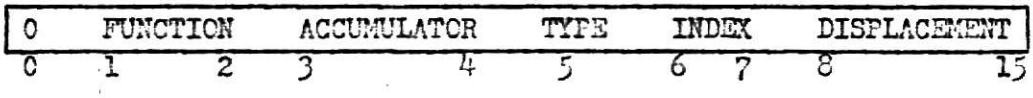
The following is a description of the structure of NCC, as given in the Nova manual (2,3).

There are four basic formats for instruction words. In all but the arithmetic and logical instructions, bit 0 if off. If bits 1 and 2 are also off, bits 3 and 4 specify the function, jump or modify memory, and the rest of the word supplies information for the calculation of effective address. Bits 8 to 15 are the displacement, bits 6 and 7 specify the index register if any, and bit 5 indicates indirection of address.



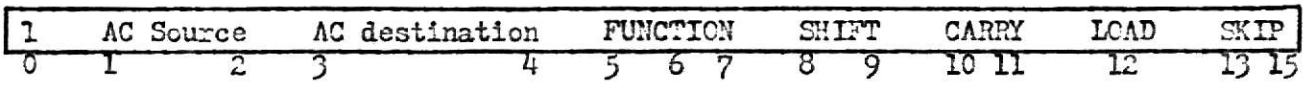
Jump and Modify Memory Instructions

If bits 1 and 2 of the instruction word are different, they specify a Move Data function. Bits 3 and 4, in this case, address an accumulator, the rest of the word is as above.



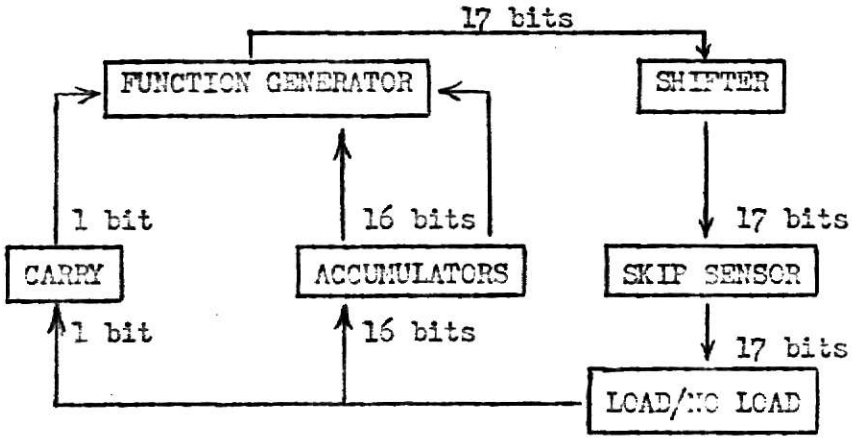
Move Data Instructions

If bit 0 of the instruction word is one, bits 5 to 7 specify an arithmetic or logical function. The structure of an arithmetic or logical instruction word is shown below.



Arithmetic and Logical Instructions

Each instruction in this class specifies one or two accumulators to supply operands to the function generator, which performs the function specified and produces a carry bit, whose value depends upon a base value specified by the instruction, the function performed and the result obtained. The base value may be derived from the carry flag or the instruction may specify an independent value.



Organization Of Arithmetic Unit

The 17-bit output of the function generator, comprising the carry bit and the 16-bit function result, then goes to the shifter. Here the 17-bit result can be rotated one place right or left, or the two 8-bit halves of the 16-bit function result can be swapped without affecting the carry bit. The 17-bit shifter output can then be tested for skip; the skip sensor can test whether the carry bit or the rest of the 17-bit word is or is not equal to zero. The 17-bit shifted word can be loaded into CARRY and one of the accumulators selected by the instruction. However, loading is not necessary; an instruction can perform a complicated arithmetic and shifting operation and test the result for skip without affecting CARRY or any accumulator.

The carry flag can be used in conjunction with the sign of the result to

detect overflow in operations on signed numbers, but its primary use is as a carry out of the most significant bit in operations on unsigned numbers, such as the lower order parts in multiple precision arithmetic. For unsigned numbers a carry is produced if addition or incrementing increases the number beyond /(2**16)-1/. In subtraction, the condition is the same if, instead of subtracting, the complement of the subtrahend is added and one is added to the result (subtraction is performed by adding the two's complement). In terms of the original operands, the subtraction (A-B) produces a carry if A is greater than, or equal to B.

SIMULATION OF THE NOVA OBJECT CODE

1. Memory Reference Instructions:

The basic logic of the simulation of memory reference instructions consists of two basic steps, the calculation of the effective memory address and the simulation of the basic operation as specified by the object code.

The calculation of simulated address is taken care by an internal procedure called ADRSR.

The ADRSR fetches bits 9 to 16 of the NOC and converts them into a binary integer called M, by appending 8 binary zeros at the left end of M. Now if the NOC specifies absolute addressing, M is saved in a field called ADDRS. As in the other two types of addressing viz, relative and indexed, the displacement is signed, the ADRSR checks bit 9 of the displacement field to see if the displacement is negative. If so, the 8 zeros in the left half of M are replaced by 8 binary ones, in order to make the integer representation negative (2's complement notation).

If the object code specifies relative addressing, this binary integer is added to the contents of the simulated program-counter and placed in ADDRS. If the address is indexed, the bits representing the index register are picked up from the NOC and are used as a subscript to point to one of the four simulated accumulators. The contents of this accumulator are converted to a binary integer by using the UNSPEC function, and then added to M. The sum is then placed in ADDRS.

Bit 6 of the object code is now checked to see if the effective address is direct or indirect. If it is an indirect address the following procedure is followed:

The rightmost 15 bits of the simulated core location, pointed to by the

contents of ADDRS, are picked up and converted into a binary integer called M, appending a zero in the high order bit of M by using the UNSPEC function. If the core location from which the 15 bits have been fetched lies between 16_{10} to 23_{10} of the simulated core, M is increased by one and the rightmost 15 bits of the updated contents are stored back into the location from which they were fetched; otherwise if the address of the simulated core location lies between 24_{10} to 31_{10}, the contents of M are decreased by one and the rightmost 15 bits of the updated contents are stored back into the location from which they were fetched. Bit 1 of this location is now fetched and if it is a one, indicating an indirect addressing, bits 2 to 15 are again converted back to a binary number by appending 8 zeros at the high order position. This number is stored in ADDRS and the cycle is repeated until such contents are fetched whose bit one is nonzero, which indicates a direct address. As soon as a direct address if found out it is stored into ADDRS. The internal procedure now checks whether the address lies within the bounds of the object module and if not, a message is printed out just as a warning and no other action is taken. It should be noted that this last operation does not represent an equivalent operation of the Nova processor. The Nova processor does not perform any such checking. It simply uses the computed address as the effective address.

At this point the internal procedure returns the control back to the main procedure, with the effective address in the field called ADDRS.

Once the simulated address has been computed only the basic operation of the NOC remains to be simulated. The following is the description of the basic operation of different memory reference instructions:

MOVE DATA INSTRUCTIONS: If the bits 1 to 3 of the NOC are 001 then the

transfer of data from memory to a register is specified, otherwise if these bits are 010 the transfer of data is the other way around, that is, from one of the registers to memory. In these two cases bits 3 and 4 of the NCC are fetched from the NCC and converted to a binary number which is used as a subscript to point to the appropriate simulated accumulator, the contents of ADDRS are used as a subscript to point to the appropriate locations in the simulated Nova-memory. The transfer of data operation is simulated by PL/1 assignment statement.

JMP & JSR INSTRUCTIONS: These instructions specify that a branch is to be made to the effective address, that is, from the memory location pointed to by the contents of ADDRS. The JMP instruction is simulated by simply replacing the contents of PC by the effective address. JSR is slightly different in that before a branch is made return address is saved in accumulator 3. This operation is simulated by:

- Increasing the program counter (PC) by 1.
- Storing the bit representation of the PC in simulated accumulator 3.
- Replacing the contents of PC by those of ADDRS.

After any of these branch instructions has been simulated, the next instruction to be simulated is fetched from the core location pointed to by the PC.

Modify Memory Instructions: If bits 4 and 5 of the NCC are 10 or 11, it specifies Increment and Skip if Zero (ISZ) or Decrement and Skip if Zero (DSZ) instructions, respectively. Both of these instructions are simulated by:

- Fetching 16 bits from the simulated core location, pointed to by the contents of ADDRS.

- Converting these 16 bits to a binary number by using the UNSPEC function.
- Depending on bits 4 and 5 of the NOC, incrementing or decrementing this binary number by one.
- Storing the bit representation of the updated binary number back into the core location from where it was originally fetched.
- Incrementing the simulated PC by 1, if the resulting binary number was zero.

The last operation causes the skipping of the next instruction in the immediately following execution cycle.

2. Arithmetic and Logic Instructions:

As described earlier, each of these instructions specifies one or two accumulators to supply operands to the function generator, which performs the function specified by the instruction. The shifter is simulated as a 17-bit field. The four accumulators are simulated by an array of four elements each of which is 16 bits.

After simulating the basic function of the instruction, the result is stored in rightmost 16 bits of SHIFTER. In Nova, the function generator produces the carry bit whose value depends upon three quantities:

1. Base value specified by the instruction.
2. The function performed.
3. The result obtained.

The carry bit is simulated by the first of the 17 bits of the SHIFTER. In the Nova the base value of the carry may be derived from the carry flag. The carry flag is simulated as a 1-bit field, called CARRY.

In the Nova the 16-bit result and a 1-bit carry are then moved to the SHIFTER. The simulation of the SHIFTER was explained previously. In the Nova environment, the 17-bit result in the SHIFTER can be rotated one place right or left, or the two 8-bit halves of the 16-bit function result can be swapped without affecting the carry bit. As the first two operations deal with the individual bits, they are simulated by using the SUBSTR function. The swapping operation is simulated by using an 8-bit intermediate work area called STORE8.

In the Nova, after the shifting operation has been performed, the skip sensor can test whether the carry bit or the other 16 bits are, or are not, equal to zero. This operation is simulated by converting the rightmost 16 bits of SHIFTER into a binary number and then comparing this number with zero.

Finally, the appropriate bit of the instruction is tested to find out whether the result of the operation is to be loaded into the destination accumulator (ACD) or not. Loading is simulated by a PL/1 assignment statement.

Simulation of Base Value of Carry Bit: The base value of the carry bit is simulated as follows:

If bits 11 & 12 of the NOC are 00, the CARRY is unchanged.

If bits 11 & 12 of the NOC are 01, the CARRY is equal to '0'B.

If bits 11 & 12 of the NOC are 10, the CARRY is equal to '1'B.

If bits 11 & 12 of the NOC are 11, the CARRY is equal to -CARRY.

Simulation of Left Rotation: This operation is simulated by:

- Storing the first bit of SHIFTER in a 1-bit field called STORE.
- Moving bits 2 to 17 into bit positions 1 to 16.
- Storing the bit in STORE into the 17th bit of SHIFTER.

Simulation of Right Rotation: This operation is simulated by:

- Storing the 17th bit of SHIFTER into a one bit field called STORE.
- Moving bits 1 to 16 into bit positions 2 to 17.
- Storing the bit in STORE in the first bit position of SHIFTER.

Simulation of Swapping: This operation is simulated by:

- Storing 2nd to 9th bits of SHIFTER in a 8-bit field called STORE8.
- Moving 10th to 17th bits of SHIFTER to 2nd to 9th bits of SHIFTER.
- Moving the 8 bits in STORE8 to the 8-bit right hand side subfield of SHIFTER.

Simulation of Loading: This operation is simulated by:

- Moving 2nd to 16th bits of SHIFTER to the Kth element of the array of registers simulating the Nova accumulators; where K is a binary number which is constructed by fetching bits 4 & 5 of the NOC (indicator of destination accumulator), and converting them into a binary number by using the UNSPEC function.
- Copying the first bit of SHIFTER into the one bit field called CARRY.

Simulation of Skipping: This operation is simulated by adding 1 to the simulated PC.

This operation causes the simulator to skip one instruction in the next execution cycle. The following text explains the simulation of various options of the Skip operation.

a) Skip on Zero Carry (SZC):

The first bit of SHIFTER is compared with a zero bit. If the comparison is equal, the skip operation is simulated.

b) Skip on Nonzero Carry (SNZ):

The first bit of SHIFTER is compared with a zero bit. If the comparison is unequal, the skip operation is simulated.

c) Skip on Zero Result (SZR):

Bits 2 to 16 of SHIFTER are converted to a binary number by UNSPEC function. This number is then compared with zero and if it is equal, the skip operation is performed.

d) Skip on Nonzero Result (SNR):

Bits 2 to 16 of SHIFTER are converted to a binary number by the UNSPEC function. This number is then compared with zero, and if it is unequal, the skip operation is performed.

e) Skip if Either Carry or Result is Zero (SEZ):

Both the comparisons of SZC and SZR are performed and if either of them shows an equal result, the skip operation is performed.

f) Skip is Both Carry and Result are Nonzero (SBN):

Both the comparisons of SZC and SZR are performed and if neither of them shows an unequal result the skip operation is performed.

g) Always Skip (SKP):

No checking is done and the PC is incremented by 1 under all the circumstances.

Simulation of the Basic Arithmetic & Logic Operations: The following text illustrates the simulation of the basic arithmetic and logic functions as specified by the NOC.

As these instructions refer to accumulators, bits 2-3 and 4-5 specify source and destination accumulators, respectively. As soon as any of these instructions is encountered these bits are picked up and converted to two binary numbers, J and K. These numbers are used as a pointer to locate the

corresponding simulated accumulators. J points to the source accumulator while K points to the destination accumulator.

a) Complement instruction (CCM):

Using the logical NOT function of PL/1, the 16 bits of Jth simulated accumulator, /REG(J)/, are logically complemented and stored in bits 2-17 of SHIFTER.

b) Negate instruction (NEG):

Using the logical NOT function of PL/1, the 16 bits of REG(J) are logically complemented. Logical complement is then converted to a binary number and 1 is added to this number in order to get the 2's complement of the original value of REG(J). Bit representation of this 2's complement is then stored in bits 2-17 of SHIFTER. Now, if bits 2-16 are all off, bit 1 is replaced by the complement of CARRY, otherwise it is replaced by CARRY.

c) Move instruction (MOV):

The 16 bits of REG(J) are stored in bits 2-17 of SHIFTER. Bit 1 of SHIFTER is then replaced by CARRY.

d) Increment instruction (INC):

The 16 bits of REG(J) are converted to a binary number. 1 is then added to this binary number and the resultant value is stored in bits 2-17 of SHIFTER, using the UNSPEC function. Now, if all the 16 bits of REG(J) were zeros, bit 1 of SHIFTER is replaced by the complement of CARRY, otherwise it is replaced by CARRY.

e) Add Complement instruction (ADC):

The 16 bits of REG(J) are logically complemented by using the logical NOT function. This complement is then converted to a binary number M, using the UNSPEC function. The 16 bits of REG(K) are now converted to another binary

number called N. M and N are now added together and the sum is stored in bits 2-17 of SHIFTER.

If REG(J) is greater than REG(K), bit 1 of SHIFTER is replaced by the complement of CARRY, otherwise it is replaced by CARRY.

f) Subtract instruction (SUB):

The simulation of this operation is performed as follows:

- REG(J) is logically complemented by using the PL/1 logical NOT function.
- This complement is then converted into a binary number, M, by using the UNSPEC function.
- 1 is added to M to get 2's complement of the original value of REG(J).
- REG(K) is then converted to another binary number, called N, by using the UNSPEC function.
- M and N are added together and the sum is placed in bits 2-17 of SHIFTER, by using the UNSPEC function.
- If original value of REG(J) was not less than that of the original value of REG(K), bit 1 of SHIFTER is replaced by the complement of CARRY; otherwise it is replaced by CARRY.

g) Add instruction (ADD):

Basic operation of this instruction is simulated as follows:

- The 16 bits of REG(J) are converted to a binary number, M, by using the UNSPEC function.
- The 16 bits of REG(K) are converted to another binary number, N, using the UNSPEC function.
- The sum of M and N is placed in bits 2-17 of SHIFTER.

• If the sum if greater than 2**16, bit 1 of SHIFTER is replaced by the complement of CARRY; otherwise it is replaced by CARRY.

h) Logical AND instruction (AND):

This operation is simulated by storing the logical AND of REG(J) and REG(K) in bits 2-17 of SHIFTER by using the PL/1 AND operator.

TESTING AND DEBUGGING OF THE SIMULATOR

This simulator has been thoroughly tested for most of the possible errors. The following method was adopted in investigating whether the simulator simulated the Nova properly:

1. The simulator was broken up into different parts depending upon the types of instructions it simulates:

   a) Move Data Instructions part.

   b) Modify Memory & Jump instructions part.

   c) Arithmetic & Logic Instructions part.

2. The calculation of memory addresses in memory reference instructions was applicable to a number of instructions, therefore it was also done in a separate module, although it was always tested along with the memory reference instructions.

3. Each of these modules was tested for most of the cases first independently of each other and then as one big module.

4. Finally, the modules for different types of instructions were merged into one module. In other words, the simulator now consists of three entities, arithmetic/logic simulation, memory reference simulation and effective address calculation simulation.

The sample programs which were tested on this simulator are included in Appendix II.

Since this simulator does not print the output, we tested the programs by executing the Nova Object Code on the simulator and on the Nova and then dumping out the simulated core and checking manually whether the result were the same as produced by Nova or not.

FUTURE MODIFICATIONS

This simulator is capable of simulating only execution of Nova Object Code, excluding I/O instructions. It does not simulate the hardware of the Nova completely. For example, it cannot simulate interrupts or the manual manipulation of the memory during the execution of a program.

The simulation of Nova I/O can be very easily incorporated in this simulator. The other two operations, however, are very difficult to simulate. The reason is that interrupts and manual manipulation are both time dependent features.

The main problem in the simulation of Nova I/O on the 360 is the interaction of the 360 I/O operations of one program with such operations of another concurrently running program. In the 360 environment, especially when using a high level language like PL/1, I/O operations are handled by the operating system; while in the Nova individual programs are responsible for their own I/O. Consequently, the logic of Nova I/O is very detailed since each and every thing is to be taken care of by the problem program. One individual step of this logic is meaningless if seen after separating it from other steps. Our simulator is not capable of decoding the logic of a Nova program. It simply reads one instruction and executes it. If it were possible to decide that the instruction being executed is a part of the I/O logic, the whole set of such instructions could be bypassed, and the control transferred to the appropriate 360 I/O routine.

Another problem is the difference in the internal and external representation of characters in the Nova and the 360. The internal representation of characters is entirely different in the two machines. Hence, whenever any I/O would be simulated a translation operation would have to be performed,

and this would somewhat affect the total execution time of a Nova program on the simulator.

One possible scheme of incorporating Nova I/O in this simulator would be to perform manual editing of the Nova listing and assigning special codes to those instructions which are a part of the I/O logic. The simulator would then be able to recognize these instructions and would be able to bypass them until it finds an explicit I/O instruction, at which point it can use the standard I/O routines of the 360. It should, however, be noted that this scheme would be practically useless in situations where the execution of a program would modify any of these marked instructions.

As of now, during the simulation of the execution of a Nova program whenever the simulator encounters an I/O instruction, it calls a procedure called I_O_ROUTINE. This procedure simply bypasses the simulation of the execution of this instruction and prints out a message:

'I/O INSTRUCTION BYPASSED AT LOCATION xxxx'

If, at any time, a routine is available which can simulate the Nova I/O, it can easily be appended to this simulator. It should, however, be noted that the simulation of Nova I/O is not easy, since detailed consideration must be given to the timing problems.

USING THE SIMULATOR

As described earlier, the main input to the simulator is the Nova Object Code. However, before reading in the Nova Object Code, the simulator looks to find out the following four initialization factors:

1. Core size of the Nova machine for which the original program was written.
2. Memory address at which the loading of the program is to begin.
3. Memory Address at which the execution of the program is to begin. (This is to take care of such programs in which constants have been defined before the first executable code.)
4. Page zero location where the load address of this program is to be saved, i.e. the Restart Address.

These pieces of information should be presented in the input stream before the actual Nova Object Code, in the form of numeric constants in exactly the same sequence as they are listed above.

IMPORTANT: <u>All four of these items should be in DECIMAL and NOT in OCTAL</u>.

The main input of the simulator is the NOC in Binary Coded Octal. For example, the Nova source code:

```
ADD 1,2
```

is equivalent to:

133000_8

When coded in binary, each digit is represented by three bits except the first digit which is either a binary one or a binary zero. Hence, when this object code is to be used as input to the simulator, it should be coded as:

```
'1011011000000000'B
```

The simulator accepts input by a 'GET LIST' statement, and therefore, the

only format requirement for the input is that two binary coded NCCs should be separated by at least one blank, if there is more than one NCC on a single card, and each of them should be enclosed in quotation marks and followed by the alphabetic B. Consequently, a maximum of four NCCs can be accomodated in a single card.

After all the NCCs have been placed in the input stream, the last NCC should be followed by a non bit-string constant. This will raise a CONVERSION condition, which will be recognized by the simulator as the end of the Nova program. This is in anticipation for the situation when the simulator will be able to perform I/O also, at which stage the input to the Nova program will also have to be present in the input stream of the simulator.

It should be noted that the simulator recognizes a CONVERSION error as a valid condition indicating the end of the Nova Object Code, hence the input stream should be thoroughly checked for any undesirable non bit-string data items.

ACKNOWLEDGEMENTS

The author is indebted to Professor Thomas N. Trump for his invaluable guidance throughout the course of this work.

Thanks are also due to Dr. K. Conrow and Dr. M.A. Calhoun for their valuable suggestions.

Gratitude is expressed to Dr. Paul S. Fisher for providing machine time on the Data General Nova.

REFERENCES AND BIBLIOGRAPHY

1. Hauber, V. P., Computer Simulation -- Something For Nothing, Data Processing Magazine, Vol 13 No. 3, March 1971.

2. English, W., How To Use The Nova Computers, Data General Corporation, Southboro, Massachusetts, April 1971.

3. Blosser, P. A. and Schneider, V. B., NOVA BASIC Emulation On A Microprogrammed Minicomputer, Proceedings of the ACM SIGPLAN symposium, University of Kansas, Lawrence, January 1972.

4. IBM Corporation, IBM System 360 Operating System PL/1 Language Reference Manual, IBM Form #GC 28-8201-3.

5. IBM Corporation, IBM System 360 Operating System PL/1 Programmer's Guide, IBM Form #GC 28-6594-6.

Appendix I

PROGRAM LISTING

```
1               INTRPR: PROC OPTIONS(MAIN);                                          1    1
                /* THIS PROGRAM SIMULATES THE EXECUTION OF A NOVA PROGRAM ON THE    */
                /* IBM 360.  THE INPUT TO THE PROGRAM IS BINARY CODED OCTADECIMAL   */
                /* NOVA OBJECT CODE.  THE PROGRAM READS IN THE ENTIRE OBJECT CODE IN*/
                /* THE SIMULATED MEMORY.  THEN IT STARTS EXECUTING THE PROGRAM FROM */
                /* THE SPECIFIED LOCATION OF THE SIMULATED MEMORY.  EACH INSTRUCTION*/
                /* IS SIMULATED TO PRODUCE THE SAME EFFECT AS THAT OF EXECUTION ON  */
                /* A NOVA MACHINE.  THE PROGRAM DOES NOT SIMULATE THE I/O INSTRUC-  */
                /* TIONS; ALTHOUGH, ONLY FOR THE PURPOSE OF DEBUGGING IT IS CAPABLE */
                /* OF READING AND PRINTING NOVA PROGRAM I/O IN A VERY CRUDE MANNER. */
2                       DCL CORE_SIZE BIN FIXED (15,0),                              1    2
                        SAVE BIN FIXED (15,0),  /* WORK AREA TO SAVE THE LOAD_POINT */ 1    2
                        LOAD_POINT BIN FIXED (15,0),
                        START_POINT BIN FIXED (15,0),
                        RESTART_ADDRESS BIN FIXED (15,0);                            1    3
3                       GET LIST(CORE_SIZE,LOAD_POINT,START_POINT,RESTART_ADDRESS);  1    4
4                       SAVE=LOAD_POINT;                                             1    5
5                       PUT SKIP LIST (' ','ECHO CHECK');
6                       PUT SKIP;
7                       PUT DATA (CORE_SIZE,LOAD_POINT,START_POINT,RESTART_ADDRESS);
8                          BEGIN;                                                    2    7
9                          DCL REG(0:3) BIT(16), /* SIMULATES NOVA ACCUMULATORS */  2    8
                              NOVA (0:CORE_SIZE) BIT (16), /* SIMULATES NOVA MEMORY */ 2  8
                               SHIFTER BIT(17), /* SIMULATES NOVA 'SHIFTER' */       2    8
                           CARRY BIT(1), /* SIMULATES NOVA CARRY FLAG */             2    8
                           STORE BIT(1),                                           2    8
                          SIGNED BIT(8),                                           2    8
                            WORK BIT(16),                                          2    8
                           PC BIN FIXED (15,0),   /* SIMULATES THE PROGRAM COUNTER */ 2  8
                           ADDRESS BIN FIXED(15,0),
                           MASK BIT(32),
                           ISTR BIN FIXED (15,0),
                           MEMORY_REF(0:3) LABEL;
                /* POINTER TO SIMULATED MEMORY LOCATIONS */                          2    9
                /* CONVERSION ERROR IS RECOGNIZED AS INTENTIONAL TO INDICATE  THAT  */ 2    9
                /* NOVA OBJECT CODE HAS BEEN COMPLETELY FED AND THAT THE FOLLOWING  */ 2    9
                /* INPUT,IF ANY,IS THE INPUT TO NOVA PROGRAM                        */ 2    9
10                         ON CONVERSION                                           2    9
11                            BEGIN;                                               3   10
12                      PUT SKIP(2) LIST (' ','END OF ECHO CHECK');
13                      PUT SKIP;
14                            GO TO SIMULATE;                                      3   12
15                            END;                                                 3   13
16                         ON ENDFILE(SYSIN)                                       2   14
17                            BEGIN;                                               3   15
18                            PUT SKIP(3)LIST('EOF ENCOUNTERED ON NOVA.SYSIN. JOB HALTE 3  16
                D');                                                                3   16
19                            GO TO ENDS;                                          3   17
2                             END;                                                 3   18
                /*  SET PC TO THE ADDRESS OF MEMORY LOCATION  IMMEDIATELY PRECEEDING */ 2  19
                /*  START_POINT                                                   */ 2   19
44                         PC=START_POINT-1;                                       2   19
                /* SAVE START_POINT IN PAGE ZERO OF SIMULATED MEMORY             */ 2   20
23                         NOVA(0)=UNSPEC(START_POINT);                            2   20
                /* READ IN NOVA OBJECT CODE                                      */ 2   21
23                      START:   GET LIST(NOVA(LOAD_POINT:));                      2   21
24                               PUT SKIP DATA(NOVA(LOAD_POINT));
```

```
                /* MODIFY POINTER TO POINT TO NEXT LOCATION                    */   2  23
25                  LOAD_POINT=LOAD_POINT+1;                                       2  23
                /* CHECK IF PROGRAM SIZE EXCEEDS CORE SIZE                     */   3  24
26                  IF LOAD_POINT>CORE_SIZE-1 THEN                                 3  24
27                     DO;                                                         4  25
28                     PUT SKIP LIST('PROGRAM SIZE GREATER THAN CORE_SIZE JOB    4  26
 TERMINATED');                                                                    4  26
29                     STOP;                                                       4  27
30                     END;                                                        4  28
31                  GO TO START;                                                   2  29
                /* START SIMULATION                                            */   2  30
                /* POINT PC TO NEXT MEMORY LOCATION                            */   2  30
32              SIMULATE:  PC=PC+1;                                               2  30
                /* CHECK IF A WRONG BRANCH HAS BEEN MADE                       */   3  31
33                  IF PC=<LOAD_POINT|PC<SAVE THEN                                 3  31
34                     DO;                                                         4  32
35                     PUT SKIP LIST('INVALID INSTR. ENCOUNTERED AT  '||PC||'    4  33
- JOB HALTED');                                                                   4  33
36                     GOTO ENDS;                                                  4  34
37                     END;                                                        4  35
                /* SIMULATE HALT INSTRUCTION                                   */   3  36
38                  IF SUBSTR(NOVA(PC),1,8)='01100110'B THEN                       3  36
39                     DO;                                                         4  37
40                     PUT SKIP LIST('JOB HALTED AT LOCATION '||PC);               4  38
41                     GO TO ENDS;                                                 4  39
42                     END;                                                        4  40
                /* CHECK FOR I/O INSTRUCTION                                   */   3  41
43                  IF SUBSTR(NOVA(PC),1,3)='011'B THEN                            3  41
44                     DO;                                                         4  42
                /* SIMULATE I/O INSTRUCTION                                    */   4  43
45                     CALL I_O_ROUTINE;                                           4  43
                /* GO BACK TO FETCH NEXT INSTRUCTION                           */   4  44
46                     GO TO SIMULATE;                                             4  44
47                     END;                                                        4  45
                /* CHECK FOR ARITHMATIC/LOGIC  INSTRUCTION                     */   3  46
48                  IF SUBSTR(NOVA(PC),1,1)='1'B THEN                              3  46
49                     DO;                                                         4  47
                /* SIMULATE  ARITHMATIC/LOGIC INSTRUCTIONS                     */   4  48
50                     CALL ARITH;                                                 4  48
                /* GO BACK TO FETCH NEXT INSTRUCTION                           */   4  49
51                     GO TO SIMULATE;                                             4  49
52                     END;                                                        4  50
                /* CALCULATE EFFECTIVE ADDRESS OF MEMORY LOCATION              */   2  51
53                  CALL ADRSR;                                                    2  51
                /* IF IT IS MODIFY MEMORY OR JUMP INSTRUCTION                  */   3  53
                /* THEN GO TO THE PROPER PROCEDURE BLOCK.                      */
54                  IF SUBSTR(NOVA(PC),1,3)='000'B THEN                            3  53
55                     DO;                                                         4  54
56                        UNSPEC(INSTR)='0000000000000000'B||SUBSTR(NOVA(PC),4,
                2);
57                        GOTO MEMORY_REF(INSTR);
58                     END;
                /* IT IS NOW LDA OR STA INSTRUCTION                            */   3  76
59                  UNSPEC(K)='0000000000000000'B||SUBSTR(NOVA(PC),4,2);           2  52
60                  IF SUBSTR(NOVA(PC),3,1)='1'B THEN                              3  76
61                     DO;                                                         4  77
                /* IF LDA, MOVE THE CONTENTS OF SIMULATED MEMORY LOCATION TO THE  */   4  78
```

```
            /* SPECIFIED SIMULATED DESTINATION ACCUMULATOR                    */   4   78
62                      REG(K)=NOVA(ADDRS);                                       4   78
63                      GO TO SIMULATE;                                           4   79
64                      END;                                                      4   80
            /* OTHERWISE STORE THE SPECIFIED SIMULATED INTO THE SPECIFIED     */   3   81
            /* MEMORY LOCATION IN THE SIMULATED CORE                          */   3   81
            /* GO BACK TO FETCH THE NEXT INSTRUCTION                          */   3   81
65                    IF SUBSTR(NOVA(PC),1,3)='010'B THEN                       3   81
66                      DO;                                                     4   82
67                      NOVA(ADDRS)=REG(K);                                     4   83
68                      GO TO SIMULATE;                                         4   84
69                      END;                                                    4   85
            /* IT IS AN UNIDENTIFIABLE INSTRUCTION                            */   2   86
70                  PUT SKIP LIST('INVALID INSTRUCTION ENCOUNTERED AT LOCATION'|   2   86
            |PC);                                                               2   86
71                  GO TO ENDS;                                                 2   87
72          MEMORY_REF (0):
            /* IT IS A JUMP INSTRUCTION                                       */   6   57
            /* SET PC TO THE NEW ADDRESS AND GO BACK TO FETCH NEXT INSTRUCTION */   6   57
            /*  FROM THAT ADDRESS                                             */   6   57
                                PC=ADDRS-1;                                     6   57
73                              GO TO SIMULATE;                                 6   58
74          MEMORY_REF (1):
            /* IT IS JSR INSTRUCTION                                          */   6   62
            /* SAVE ADDRESS OF NEXT LOCATION IN SIMULATED ACCUMULATOR 3 AND  SET */ 6   62
            /* PC TO THE NEW ADDRESS                                          */   6   62
                                PC=PC+1;                                        6   62
75                              REG(3)=UNSPEC(PC);                              6   63
            /* GO BACK TO FETCH NEXT INSTRUCTION FROM NEW ADDRESS             */   6   64
76                              PC=ADDRS-1;                                     6   64
77                              GO TO SIMULATE;                                 6   65
78          MEMORY_REF (2):
            /* IT IS AN ISZ INSTRUCTION.  CONVERT THE CONTENTS OF THE MEMORY  */
            /* LOCATION TO A BINARY NUMBER.                                   */
                            UNSPEC(M)=NOVA(ADDRS);                              4   67
            /* ADD 1 TO THE BINARY NUMBER                                     */
79                              M=M+1;                                          5   69
            /* STORE THE UPDATED NUMBER BACK INTO THE MEMORY                  */
80          RTRN:               NOVA(ADDRS)=UNSPEC(M);
            /* IF THE NUMBER IS NOW ZERO,SET PC TO FETCH NEXT INSTRUCTION FROM */   5   72
            /* NEXT TO NEXT LOCATION                                          */   5   72
81                              IF M=0 THEN                                     5   72
82                              PC=PC+1;                                        5   73
83                          GO TO SIMULATE;                                     4   74
84          MEMORY_REF (3):
            /* IT IS A DSZ INSTRUCTION. CONVERT THE CONTENTS INTO A BINARY    */
            /* NUMBER AND SUBTRACT 1 FROM THIS NUMBER.                        */
                            UNSPEC(M)=NOVA(ADDRS);
85                          M=M-1;
86                          GOTO RTRN;
87          ARITH:        PROC;                                                 3   88
88                          DCL ARITH_LOGIC(0:7) LABEL;
            /* SET THE BASE VALUE FOR CARRY                                   */   4   89
89                          IF SUBSTR(NOVA(PC),11,2)='01'B THEN                 4   89
90                          CARRY='0'B;                                         4   90
91                          ELSE                                                4   91
91                              IF SUBSTR(NOVA(PC),11,2)='10'B THEN             5   91
```

```
92                          CARRY='1'B;                                    5  92
93                          ELSE                                           5  93
93                             IF SUBSTR(NOVA(PC),11,2)='11'B THEN         6  93
94                             CARRY=¬CARRY;                               6  94
            /* GET POINTERS TO THE SIMULATED SOURCE AND DESTINATION ACCUMULATORS*/  3  95
95                       UNSPEC(J)='000000000000000'B||SUBSTR(NOVA(PC),2,2);  3  95
96                       UNSPEC(K)='000000000000000'B||SUBSTR(NOVA(PC),4,2);  3  96
97                       UNSPEC(INSTR)='0000000000000'B||SUBSTR(NOVA(PC),6,3);
98                       GOTO ARITH_LOGIC (INSTR);
99          ARITH_LOGIC(0):
            /* IT IS COMPLEMENT INSTRUCTION                              */  5  99
            /* GET LOGICAL COMPLEMENT OF SOURCE ACCUMULATOR AND STORE IN SHIFTER*/  5  99
                        SUBSTR(SHIFTER,2,16)=¬(REG(J));                    5  99
            /* SUPPLY THE BASE VALUE OF CARRY AS THE CARRY BIT           */  5 100
100                     SUBSTR(SHIFTER,1,1)=CARRY;                         5 100
101                     GO TO SHIFT;                                       5 101
102         ARITH_LOGIC(1):
            /* IT IS A NEGATE INSTRUCTION                                */  5 105
            /* GET LOGICAL COMPLEMENT OF SOURCE ACCUMULATOR              */  5 105
                        UNSPEC(M)=¬REG(J);                                 5 105
            /* ADD 1 TO LOGICAL COMPLEMENT TO GET 2'S COMPLEMENT         */  5 106
103                     M=M+1;                                             5 106
            /* STORE IT IN THE SHIFTER                                   */  5 107
104                     SUBSTR(SHIFTER,2,16)=UNSPEC(M);                    5 107
105                     UNSPEC(M)=REG(J);                                  5 108
            /* IF SOURCE ACCUMULATOR IS ZERO, SUPPLY THE COMPLEMENT OF CARRY AS */  6 109
            /* THE NEW CARRY BIT                                         */  6 109
106                        IF M=0 THEN                                     6 109
107                        SUBSTR(SHIFTER,1,1)=¬CARRY;                     6 110
            /* OTHERWISE SUPPLY THE BASE VALUE OF CARRY AS THE CARRY BIT */  6 111
108                        ELSE                                            6 111
108                        SUBSTR(SHIFTER,1,1)=CARRY;                      6 111
109                     GO TO SHIFT;                                       5 112
110         ARITH_LOGIC(2):
            /* IT IS A MOVE INSTRUCTION                                  */  5 116
            /* MOVE SOURCE ACCUMULATOR TO SHIFTER AND BASE VALUE OF CARRY TO */  5 116
            /* THE CARRY BIT                                             */  5 116
                        SUBSTR(SHIFTER,2,16)=REG(J);                       5 116
111                     SUBSTR(SHIFTER,1,1)=CARRY;                         5 117
112                     GO TO SHIFT;                                       5 118
113         ARITH_LOGIC(3):
            /* IT IS AN INCREMENT INSTRUCTION                            */  5 122
            /* CONVERT SOURCE ACCUMULATOR TO A BINARY NUMBER             */  5 122
                        UNSPEC(M)=REG(J);                                  5 122
            /* ADD 1 TO THE BINARY NUMBER AND STORE IT IN THE SHIFTER    */  5 123
114                     M=M+1;                                             5 123
115                     SUBSTR(SHIFTER,2,16)=UNSPEC(M);                    5 124
116                     M=M-1;                                             5 125
            /*IF SOURCE ACCUMULATOR WAS  -1  COMPLEMENT THE BASE VALUE OF CARRY */  6 126
117                        IF M=-1 THEN                                    6 126
118                        SUBSTR(SHIFTER,1,1)=¬CARRY;                     6 127
            /* OTHERWISE SUPPLY THE BASE VALUE OF CARRY AS THE CARRY BIT */  6 128
119                        ELSE                                            6 128
119                        SUBSTR(SHIFTER,1,1)=CARRY;                      6 128
120                     GO TO SHIFT;                                       5 129
121         ARITH_LOGIC(4):
            /* IT IS AN ADC INSTRUCTION                                  */  5 133
```

```
                    /* CONVERT SOURCE ACCUMULATOR TO ITS LOGICAL COMPLEMENT          */   5  133
                    /* CONVERT DESTINATION ACCUMULATOR TO A BINARY NUMBER            */   5  133
                                   UNSPEC(M)=¬REG(J);                                    5  133
122                                UNSPEC(N)=REG(K);                                     5  134
                    /* ADD THE LOGICAL COMPLEMENT OF SOURCE ACCUMULATOR TO THE BINARY */   5  135
                    /* NUMBER                                                        */   5  135
123                                M=M+N;                                                5  135
                    /* STORE THE RESULT IN THE SHIFTER                               */   5  136
124                                SUBSTR(SHIFTER,2,16)=UNSPEC(M);                       5  136
                    /* IF SOURCE ACCUMULATOR IS GREATER THAN DESTINATION ACCUMULATOR  */   6  137
                    /* THEN SUPPLY THE COMPLEMENT OF THE BASE VALUE OF CARRY AS THE   */   6  137
                    /* CARRY BIT                                                     */   6  137
125                                   IF UNSPEC(REG(K))>UNSPEC(REG(J))THEN               6  137
126                                   SUBSTR(SHIFTER,1,1)=¬CARRY;                        6  138
                    /* OTHERWISE SUPPLY THE BASE VALUE OF CARRY AS THE CARRY BIT     */   6  139
127                                   ELSE                                               6  139
127                                   SUBSTR(SHIFTER,1,1)=CARRY;                         6  139
128                                GO TO SHIFT;                                          5  140
129                 ARITH_LOGIC(5):
                    /* IT IS A SUBTRACT INSTRUCTION                                  */   5  144
                    /* GET 2'S COMPLEMENT OF THE SOURCE ACCUMULATOR                  */   5  144
                                   UNSPEC(M)=¬REG(J);                                    5  144
130                                M=M+1;                                                5  145
                    /* CONVERT DESTINATION ACCUMULATOR TO A BINARY NUMBER             */   5  146
131                                UNSPEC(N)=REG(K);                                     5  146
                    /* ADD 2'S COMPLEMENT TO THE BINARY NUMBER                       */   5  147
                    /* STORE THE RESULT IN THE SHIFTER                               */   5  147
132                                M=M+N;                                                5  147
133                                SUBSTR(SHIFTER,2,16)=UNSPEC(M);                       5  148
                    /* IF SOURCE ACCUMULATOR NOT GREATER THAN THE DESTINATION        */   6  149
                    /* ACCUMULATOR, SUPPLY THE COMPLEMENT OF THE BASE VALUE OF CARRY */   6  149
                    /* AS THE CARRY BIT                                              */   6  149
134                                   IF REG(J)¬>REG(K)THEN                              6  149
135                                   SUBSTR(SHIFTER,1,1)=¬CARRY;                        6  150
                    /* OTHERWISE SUPPLY THE BASE VALUE OF CARRY AS THE CARRY BIT     */   6  151
136                                   ELSE                                               6  151
136                                   SUBSTR(SHIFTER,1,1)=CARRY;                         6  151
137                                GO TO SHIFT;                                          5  152
138                 ARITH_LOGIC(6):
                    /* IT IS AN ADD INSTRUCTION                                      */   4  154
                    /* CONVERT SOURCE AND DESTINATION ACCUMULATORS TO BINARY NUMBERS */   4  154
                                   UNSPEC(M)=REG(J);                                     5  156
139                                UNSPEC(N)=REG(K);                                     5  157
                    /* ADD THE BINARY NUMBERS TOGETHER                               */   5  158
                    /* STORE THE SUM IN SHIFTER                                      */   5  158
140                                M1=M+N;
141                                M=M+N;                                                5  158
142                                SUBSTR(SHIFTER,2,16)=UNSPEC(M);                       5  159
                    /* IF SUM IS NOT LESS THAN UNSIGNED (2**16), SUPPLY THE COMPLEMENT */
                    /* OF THE BASE VALUE OF CARRY AS THE CARRY BIT;                  */
143                                IF UNSPEC(M1)¬<'0000000000000010000000000000000'B
                THEN
144                                   SUBSTR(SHIFTER,1,1)=¬CARRY;                        6  162
                    /* OTHERWISE SUPPLY THE BASE VALUE OF CARRY AS THE CARRY BIT     */   6  163
145                                   ELSE                                               6  163
145                                   SUBSTR(SHIFTER,1,1)=CARRY;                         6  163
146                                GO TO SHIFT;                                          5  164
```

```
147             ARITH_LOGIC(7):
                /* IT IS AN AND INSTRUCTION                                    */  3  166
                /* 'AND' THE SOURCE AND DESTINATION ACCUMULATORS TO EACH OTHER  */  3  166
                /* STORE THE RESULT IN SHIFTER                                 */  3  166
                        SUBSTR(SHIFTER,2,16)=REG(J)&REG(K);                        3  166
                /* SUPPLY BASE VALUE OF CARRY AS THE CARRY BIT                 */  3  167
148                     SUBSTR(SHIFTER,1,1)=CARRY;                                 3  167
149             SHIFT:                                                             4  168
                /* PERFORM THE SHIFTING OF BITS IN SHIFTER AS SPECIFIED BY      */  4  168
                /* BITS 9 AND 10 OF THE INSTRUCTION                            */  4  168
                /* SIMULATION OF LEFT ROTATION                                 */  4  168
                        IF SUBSTR(NOVA(PC),9,2)='01'B THEN                         4  168
150                        DO;                                                     5  169
151                        STORE=SUBSTR(SHIFTER,1,1);                              5  170
152                        SUBSTR(SHIFTER,1,16)=SUBSTR(SHIFTER,2,16);
153                        SUBSTR(SHIFTER,17,1)=STORE;                             5  174
154                        GO TO LOAD;                                             5  175
155                        END;                                                    5  176
                /* SIMULATION OF RIGHT ROTATION                                */  4  177
156                     IF SUBSTR(NOVA(PC),9,2)='10'B THEN                         4  177
157                        DO;                                                     5  178
158                        STORE=SUBSTR(SHIFTER,17,1);                             5  179
159                        SUBSTR(SHIFTER,2,16)=SUBSTR(SHIFTER,1,16);
160                        SUBSTR(SHIFTER,1,1)=STORE;                              5  183
161                        GO TO LOAD;                                             5  184
162                        END;                                                    5  185
                /* SIMULATION OF SWAPPING                                      */  4  186
163                     IF SUBSTR(NOVA(PC),9,2)='11'B THEN                         4  186
164                        DO;                                                     5  187
165                        STORE8=SUBSTR(SHIFTER,2,8);                             5  188
166                        SUBSTR(SHIFTER,2,8)=SUBSTR(SHIFTER,10,8);               5  189
167                        SUBSTR(SHIFTER,10,8)=STORE8;                            5  190
168                        END;                                                    5  191
169             LOAD:                                                              4  192
                /* IF NO-LOAD BIT IN THE INSTRUCTION IS OFF, LOAD THE SHIFTER OUT- */  4  192
                /* PUT IN THE DESTINATION ACCUMULATOR AND CARRY BIT INTO CARRY  */  4  192
                        IF SUBSTR(NOVA(PC),13,1)='0'B THEN                         4  192
170                        DO;                                                     5  193
171                        REG(K)=SUBSTR(SHIFTER,2,16);                            5  194
172                        CARRY=SUBSTR(SHIFTER,1,1);                              5  195
173                        END;                                                    5  196
174             SKIP:                                                              4  197
                /* PERFORM CHECKING FOR DIFFERENT SKIP OPTIONS                  */  4  197
                /* SIMULATION OF SKP OPTION OF SKIPPING                        */  4  197
                        IF SUBSTR(NOVA(PC),14,3)='001'B THEN                       4  197
175                        DO;                                                     5  198
176                        PC=PC+1;                                                5  199
177                        RETURN;                                                 5  200
178                        END;                                                    5  201
                /* SIMULATION OF SZC OPTION OF SKIPPING                        */  4  202
179                     IF SUBSTR(NOVA(PC),14,3)='010'B THEN                       4  202
180                        DO;                                                     5  203
181                           IF SUBSTR(SHIFTER,1,1)='0'B THEN                     6  204
182                           PC=PC+1;                                             6  205
183                        RETURN;                                                 5  206
184                        END;                                                    5  207
                /* SIMULATION OF SNC OPTION OF SKIPPING                        */  4  208
```

```
185                         IF SUBSTR(NOVA(PC),14,3)='011'B THEN                    4  208
186                            DO;                                                  5  209
187                               IF SUBSTR(SHIFTER,1,1)='1'B THEN                  6  210
188                                  PC=PC+1;                                       6  211
189                            RETURN;                                              5  212
190                            END;                                                 5  213
191                      UNSPEC(M)=SUBSTR(SHIFTER,2,16);                            3  214
         /* SIMULATION OF SZR OPTION OF SKIPPING                                */  4  215
192                         IF SUBSTR(NOVA(PC),14,3)='100'B THEN                    4  215
193                            DO;                                                  5  216
194                               IF M=0 THEN                                       6  217
195                                  PC=PC+1;                                       6  218
196                            RETURN;                                              5  219
197                            END;                                                 5  220
         /* SIMULATION OF SNR OPTION OF SKIPPING                                */  4  221
198                         IF SUBSTR(NOVA(PC),14,3)='101'B THEN                    4  221
199                            DO;                                                  5  222
200                               IF M¬=0 THEN                                      6  223
201                                  PC=PC+1;                                       6  224
202                            RETURN;                                              5  225
203                            END;                                                 5  226
         /* SIMULATION OF SEC OPTION OF SKIPPING                                */  4  227
204                         IF SUBSTR(NOVA(PC),14,3)='110'B THEN                    4  227
205                            DO;                                                  5  228
206                               IF(M=0)|(SUBSTR(SHIFTER,1,1)='0'B)THEN            6  229
207                                  PC=PC+1;                                       6  230
208                            RETURN;                                              5  231
209                            END;                                                 5  232
         /* SIMULATION OF SBN OPTION OF SKIPPING                                */  4  233
210                         IF SUBSTR(NOVA(PC),14,3)='111'B THEN                    4  233
211                            DO;                                                  5  234
212                               IF(M¬=0)&(SUBSTR(SHIFTER,1,1)¬='0'B)THEN          6  235
213                                  PC=PC+1;                                       6  236
214                            RETURN;                                              5  237
215                            END;                                                 5  238
216                      END ARITH;                                                 3  239
217            ADRSR:    PROC;                                                     3  240
         /* CONVERT DISPLACEMENT TO BINARY NUMBER                               */  3  241
218                      UNSPEC(M)='000000000'B||SUBSTR(NOVA(PC),9,8);              3  241
         /* IF ABSOLUTE ADDRESSING,USE DISPLACEMENT AS EFFECTIVE ADDRESS        */  4  242
219                         IF SUBSTR(NOVA(PC),7,2)='00'B THEN                      4  242
220                            DO;                                                  5  243
221                            ADDRS=M;                                             5  244
         /* AND GO TO CHECK ADDRESS TYPE                                        */  5  245
222                            GO TO TYPE;                                          5  245
223                            END;                                                 5  246
         /* IF THE DISPLACEMENT IS NEGATIVE,IT IS IN 2'S COMPLEMENT             */  4  247
224                         IF SUBSTR(NOVA(PC),9,1)='1'B THEN                       4  247
225                         UNSPEC(M)='111111111'B||SUBSTR(NOVA(PC),9,8);           4  248
         /* IF RELATIVE ADDRESSING,ADD DISPLACEMENT AND PC TO GET EFFECTIVE     */  4  249
         /* ADDRESS                                                             */  4  249
226                         IF SUBSTR(NOVA(PC),7,2)='01'B THEN                      4  249
227                            DO;                                                  5  250
228                            ADDRS=PC+M;                                          5  251
229                            GO TO TYPE;                                          5  252
230                            END;                                                 5  253
         /* IF INDEXED ADDRESSING ,ADD INDEX REGISTER & DISPLACEMENT TO GET     */  4  254
```

```
                /* EFFECTIVE ADDRESS                                          */   4  254
231                     IF SUBSTR(NOVA(PC),7,2)='10'B THEN                        4  254
232                     UNSPEC(N)=REG(2);                                         4  255
233                     ELSE                                                      4  256
233                     UNSPEC(N)=REG(3);                                         4  256
234                   ADDRS=M+N;                                                  3  257
                /* IF INDIRECT ADDRESSING                                     */   4  258
                /* FETCH THE CONTENTS OF EFFECTIVE ADDRESS                    */   4  258
235             TYPE:   IF SUBSTR(NOVA(PC),6,1)='1'B THEN                         4  258
236                        DO;                                                    5  259
237             BACK:      UNSPEC(M)='0'B||SUBSTR(NOVA(ADDRS),2,15);              5  260
                /* CHECK FOR AUTOINDEXING                                     */   6  261
                /* AUTO-INCREMENTING SIMULATION                               */   6  261
238                           IF ADDRS¬<16&ADDRS¬>23 THEN                         6  261
239                              DO;                                              7  262
240                              M=M+1;                                           7  263
241                              WORK=UNSPEC(M);                                  7  264
242                              SUBSTR(NOVA(ADDRS),2,15)=SUBSTR(WORK,2,15);
243                              END;                                             7  266
                /* AUTO-DECREMENTING SIMULATION                               */   6  267
244                           ELSE                                                6  267
244                              IF ADDRS¬<24&ADDRS¬>31 THEN                      7  267
245                                 DO;                                           8  268
246                                 M=M-1;                                        8  269
247                                 WORK=UNSPEC(M);                               8  270
248                              SUBSTR(NOVA(ADDRS),2,15)=SUBSTR(WORK,2,15);
249                                 END;                                          8  272
                /* IF THE CONTENTS AGAIN SPECIFY INDIRECT ADDRESSING, REPEAT THE */   6  273
                /* CYCLE USING THE CONTENTS OF EFFECTIVE ADDRESS AS NEW ADDRESS */   6  273
250                           IF SUBSTR(NOVA(ADDRS),1,1)='1'B THEN                6  273
251                              DO;                                              7  274
252                              UNSPEC(ADDRS)='0'B||SUBSTR(NOVA(ADDRS),2,15);    7  275
253                              GO TO BACK;                                      7  276
254                              END;                                             7  277
                /*OTHERWISE USE CONTENTS AS THE FINAL EFFECTIVE ADDRESS      */   5  278
255                        ADDRS=M;                                               5  278
256                        END;                                                   5  279
                /* IF COMPUTED ADDRESS NOT WITHIN THE SCOPE OF OBJECT MODULE, */   4  280
                /* PRINT OUT A WARNING MESSAGE                                */   4  280
257                     IF ADDRS¬<LOAD_POINT|ADDRS<SAVE THEN                      4  280
258                     PUT SKIP LIST('OUT_OF_PROGRAM_ADDRESS REFERENCE AT INS    4  281
TRUCTION'||PC);                                                                   4  281
259                   END ADRSR;                                                  3  282
260             I_O_ROUTINE:  PROC;                                               3  283
261                     IF NOVA(PC)='011000000010001000'B THEN                    4  284
262                     RETURN;                                                   4  285
263                     IF NOVA(PC)='011001110100100'B|NOVA(PC)='011001111000     4  286
1010'B THEN                                                                       4  286
264                        DO PC=PC+1;                                            5  287
265                        RETURN;                                                5  288
266                        END;                                                   5  289
267                     IF NOVA(PC)='011010100010001001'B THEN                    4  290
268                        DO;                                                    5  291
269                        PUT LIST(REG(1));                                      5  292
270                        RETURN;                                                5  293
271                        END;                                                   5  294
272                     IF NOVA(PC)='011100100010001001'B THEN                    4  295
```

```
273                                  DO;                                          5  296
274                                  PUT LIST(REG(2));                            5  297
275                                  RETURN;                                      5  298
276                                  END;                                         5  299
277                               IF NOVA(PC)='0111000101001000'B THEN            4  300
278                                  DO;                                          5  301
279                                  GET LIST(REG(2));                            5  302
280                                  RETURN;                                      5  303
281                                  END;                                         5  304
282                            PUT SKIP LIST('I/O INSTRUCTION BEING BYPASSED AT LOCATION
               '||PC);                                                          3  305
283                            END I_O_ROUTINE;                                 3  306
284                     ENDS:                                                   2  307
                        /* PRINT OUT THE SIMULATED ACCUMULATORS, SHIFTER, CARRY, AND ALL THE*/  2  307
                        /* MEMORY                                                */  2  307
                                 PUT SKIP(2)DATA(PC);                           2  307
285                              PUT SKIP(2)DATA(REG);                          2  308
286                              PUT SKIP(2)DATA(SHIFTER,CARRY);                2  309
287                              PUT SKIP;                                      2  310
288                                 DO K=SAVE TO LOAD_POINT-1;                  3  311
289                                 PUT DATA(NOVA(K));                          3  312
290                                 END;                                        3  313
291                              END;                                           2  314
292                     END INTRPR;                                             1  315
```

Appendix II

TESTING RUNS

```
                    ECHO CHECK
CORE_SIZE=     4096     LOAD_POINT=      256     START_POINT=      256   RESTART_ADDRESS=       32;
NOVA(256)='001C0001CCCC1001'B;                          LDA   0,CNT-2
NOVA(257)='0100000CCCCC0001'B;                          STA   0,21
NOVA(258)='0010000100001000'B;                          LDA   0,CNT-1
NOVA(259)='0100000CCCC11101'B;                          STA   0,35
NOVA(260)='0010010CCC010001'B;                LOOP:     LDA   0,@21
NOVA(261)='0100010000011101'B;                          STA   0,@35
NOVA(262)='0001100ICCCC0101'B;                          DSZ   CNT
NOVA(263)='0000000111111101'B;                          JMP   LOOP
NOVA(264)='0110011000111111'B;                          HALT
NOVA(265)='0000000100001011'B;                          TAB1-1
NOVA(266)='0000000ICCC10110'B;                          TAB2
NOVA(267)='0000000000000101'B;                CNT:      5
NOVA(268)='0000000C1C000001'B;                TAB1:     .TXT "A"
NOVA(269)='0C0000001C100010'B;                          .TXT "B"
NOVA(270)='0000000C1CC00011'B;                          .TXT "C"
NOVA(271)='0000000001C00100'B;                          .TXT "D"
NOVA(272)='0C0000001C1C0101'B;                          .TXT "E"
NOVA(273)='0000000CCC1C00CC'B;                          .TXT " "
NOVA(274)='0000000000100000'B;                          .TXT " "
NOVA(275)='C000000CCC1C0000'B;                          .TXT " "
NOVA(276)='0000000CCC1C0000'B;                          .TXT " "
NOVA(277)='0000000000100000'B;                          .TXT " "
NOVA(278)='0C0000000C100000'B;                TAB2:     .TXT " "
                                                        .END
                    END OF ECHO CHECK

OLT_OF_PROGRAM_ADDRESS REFERENCE AT INSTRUCTION      257
OLT_OF_PROGRAM_ADDRESS REFERENCE AT INSTRUCTION      259
JCB HALTED AT LOCATION       264

PC=      264;

REG(0)='00C0000001000101'B                  REG(1)='0111C01101101010'B                  REG(2)='0100100000100000
'B                  REG(3)='0111010101111110'B;

SHIFTER='0001101001110010'B                 CARRY='0'B;
NOVA(256)='0C1000010CCC01001'B;             NOVA(257)='0100CCCCCCC010CC1'B;             NOVA(258)='0010000100001
000'B;              NOVA(259)='01C00CCCCCC011101'B;             NOVA(260)='0010010000010001'B;
NOVA(261)='0100010CCCC11101'B;              NOVA(262)='0001100ICC000101'B;              NOVA(263)='0000000111111
101'B;              NOVA(264)='0110011000111111'B;              NOVA(265)='0000000100001011'B;
NOVA(266)='0C0000001CCC10110'B;             NOVA(267)='0000000000000000'B;              NOVA(268)='0000000001000
001'B;              NOVA(269)='0000000CC10C0010'B;              NOVA(270)='0000000001000011'B;
NOVA(271)='0000001000100100'B;              NOVA(272)='0C0000C0C01000101'B;             NOVA(273)='0000000001000
101'B;              NOVA(274)='0000000C01000100'B;              NOVA(275)='0000000001000011'B;
NOVA(276)='0000000CC1C00010'B;              NOVA(277)='0000000001000001'B;              NOVA(278)='0000000000100
000'B;
```

```
                        ECHO CHECK
CORE_SIZE=     4096     LOAD_POINT=       0     START_POINT=       0   RESTART_ADDRESS=      256;
NOVA(0)='0000100000000001'B;                        JSR    ABC
NOVA(1)='0101100000000101'B;                ABC:    STA    3,AR
NOVA(2)='0001100000000110'B;                        DSZ    ONE
NOVA(3)='0000100000000101'B;                        JMP    @AB
NOVA(4)='0110011000111111'B;                        HALT
NOVA(5)='0000000000000000'B;                AR:     0
NOVA(6)='0000000000001000'B;                ONE:    10
                                                    .END
                        END OF ECHO CHECK

JOB HALTED AT LOCATION          4

PC=         4;

REG(0)='0100010000010000'B                  REG(1)='0111010110101010'B                  REG(2)='0100100000100000
'B                          REG(3)='0000000000000001'B;

SHIFTER='0001010001110010'B                 CARRY='0'B;
NOVA(0)='0000100000000001'B;                NOVA(1)='0101100000000101'B;                NOVA(2)='0001100000000011
0'B;                        NOVA(3)='0000010000000101'B;                NOVA(4)='0110011000111111'B;
NOVA(5)='0000000000000001'B;                NOVA(6)='0000000000000000'B;
```

```
                        ECHO CHECK
CORE_SIZE=     4096     LOAD_POINT=        0     START_POINT=        0     RESTART_ADDRESS=      256;
NOVA(0)='0010100C0CCC1010'B;                    LDA   1,A
NOVA(1)='1010101011000000'B;                    MOVS  1,1
NOVA(2)='0011000000001011'B;                    LDA   2,B
NOVA(3)='1100111000000000'B;                    ADD   2,1
NOVA(4)='0100100000001101'B;                    STA   1,D
NOVA(5)='1101C01C11CCCC00'B;                    MOVS  2,2
NOVA(6)='0010100000C01100'B;                    LDA   1,C
NOVA(7)='1100111000000000'B;                    ADD   2,1
NOVA(8)='0100100000001110'B;                    STA   1,E
NOVA(9)='0110011000111111'B;                    HALT
NOVA(10)='0000000001000001'B;          A:       .TXT  "A"
NOVA(11)='0000000010000010'B;          B:       .TXT  "B"
NOVA(12)='0000000001000011'B;          C:       .TXT  "C"
NOVA(13)='0000000000100000'B;          D:       .TXT  " "
NOVA(14)='0000000000100000'B;          E:       .TXT  " "
                                                .END
                        END OF ECHO CHECK

JOB HALTED AT LOCATION        9

PC=        9;

REG(0)='0100010000010000'B                      REG(1)='0100001001000011'B                      REG(2)='0100001000000000
'B                      REG(3)='0111010101111110'B;

SHIFTER='0010000100100011'B                     CARRY='0'B;
NOVA(0)='0010100000001010'B;                    NOVA(1)='1010101011000000'B;                    NOVA(2)='0011000000000101
1'B;                    NOVA(3)='1100111000000000'B;                    NOVA(4)='0100100000001101'B;
NOVA(5)='1101001011000000'B;                    NOVA(6)='0010100000001100'B;                    NOVA(7)='1100111000000000
0'B;                    NOVA(8)='0100100000001110'B;                    NOVA(9)='0110011000111111'B;
NOVA(10)='0000000001000001'B;                   NOVA(11)='0000000001000010'B;                   NOVA(12)='0000000001000000
11'B;                   NOVA(13)='0100000101000010'B;                   NOVA(14)='0100001001000011'B;
```

```
                    ECHO CHECK
CORE_SIZE=     4096    LOAD_POINT=        0    START_POINT=        0    RESTART_ADDRESS=      256;
NOVA(0)='0011000000001011'B;                        LDA   2,T1
NOVA(1)='0011100000001100'B;                        LDA   3,T2
NOVA(2)='0001000100000000'B;              AGAIN:    LDA   0,0,2
NOVA(3)='0100001100000000'B;                        STA   0,0,3
NOVA(4)='0001100000001010'B;                        DSZ   LMT
NOVA(5)='0000000000000111'B;                        JMP   INCRE
NOVA(6)='0110011000111111'B;                        HALT
NOVA(7)='1101001100000000'B;              INCRE:    INC   2,2
NOVA(8)='1111011000000000'B;                        INC   3,3
NOVA(9)='0000000111111001'B;                        JMP   AGAIN
NOVA(10)='0000000000000101'B;             LMT:      5
NOVA(11)='0000000000001101'B;             T1:       TA
NOVA(12)='0000000000010010'B;             T2:       TB
NOVA(13)='0000000001000001'B;             TA:       .TXT  "A"
NOVA(14)='0000000001000010'B;                       .TXT  "B"
NOVA(15)='0000000001000011'B;                       .TXT  "C"
NOVA(16)='0000000001000100'B;                       .TXT  "D"
NOVA(17)='0000000001000101'B;                       .TXT  "E"
NOVA(18)='0000000000100000'B;             TB:       .TXT  " "
NOVA(19)='0000000000100000'B;                       .TXT  " "
NOVA(20)='0000000000100000'B;                       .TXT  " "
NOVA(21)='0000000000100000'B;                       .TXT  " "
NOVA(22)='0000000000100000'B;                       .END

                    END OF ECHO CHECK

JOB HALTED AT LOCATION         6

PC=        6;

REG(0)='0000000001000101'B                  REG(1)='1001000100100000'B                  REG(2)='0000000000010001
'B                  REG(3)='0000000000010110'B;

SHIFTER='0000000000010110'B                 CARRY='0'B;
NOVA(0)='0011000000001011'B;                NOVA(1)='0011100000001100'B;                NOVA(2)='0010001000000000
0'B;                NOVA(3)='0100001100000000'B;                NOVA(4)='0001100000001010'B;
NOVA(5)='0000000000000111'B;                NOVA(6)='0110011000111111'B;                NOVA(7)='1101001100000000
0'B;                NOVA(8)='1111011000000000'B;                NOVA(9)='0000000111111001'B;
NOVA(10)='0000000000000000'B;               NOVA(11)='0000000000001101'B;               NOVA(12)='0000000000010010
0'B;                NOVA(13)='0000000001000001'B;               NOVA(14)='0000000001000010'B;
NOVA(15)='0000000001000011'B;               NOVA(16)='0000000001000100'B;               NOVA(17)='0000000001000101
01'B;               NOVA(18)='0000000001000001'B;               NOVA(19)='0000000001000010'B;
NOVA(20)='0000000001000011'B;               NOVA(21)='0000000001000100'B;               NOVA(22)='0000000001000101
01'B;
```

```
ECHO CHECK
'000100000000100'B                    LOOP:   ISZ     ONE
'000110000000101'B                            DSZ     TWO
'000000000000010'B                            JMP     LOOP
'011001100111111'B                            HALT
'000000000000000'B                    ONE:    0
'000000100000000'B                    TWO:    1000
END OF ECHO CHECK                             .END
JOB HALTED AT LOCATION        3

PC=       3;

REG(0)='0000000000000001'B                    REG(1)='0001101101000010'B                    REG(2)='0100000101000100
'B                    REG(3)='0000000000000001'B;

SHIFTER='0010000000011000'B                   CARRY='0'B;
NOVA(0)='0001000000000100'B;                  NOVA(1)='0001100000000101'B;                  NOVA(2)='0000000000000000
0'B;                  NOVA(3)='0110011000111111'B;                  NOVA(4)='0000001000000000'B;
NOVA(5)='0000000000000000'B;
```

```
---
        102432                  .DALC SG=SUB;# 0,0,5,C-
        000037                  .LOC 37         ; RESTART ADDR. 37
00037   000420                  START
        000400                  .LOC 400
00400   000012 LF:      12      ;LINE FEED
00401   000015 CR:      15      ;CARR RET.
00402   000044 STR:     44      ; $
00403   000007 CM:      7
00404   000017 MK:      17
00405   000000 SAV:     0
00406   000000          0
00407   000000          0
00410   177777 F:       -1
00411   177772 X:       -6
00412   000060 SX:      60
00413   000000 SV:      0
00414   000041 A:       41
00415   000011 B:       11
00416   000040 BLK:     40
00417   000007 BEL:     7
00420   024760 START:   LDA 1,STR
00421   004520          JSR PRT
00422   060113          NIOS TTI
00423   102400          SUB 0,0
00424   024771 LP:      LDA 1,B
00425   004520          JSR RED ; GET CHARACTER
00426   034756          LDA 3,MK
00427   173400          AND 3,2
00430   132432          SG 1,2          ; IF CHAR FOUND END OF INP STR
00431   000406          JMP DN          ; GO TO DN
00432   101120          MOVZL 0,0
00433   101120          MOVZL 0,0
00434   101120          MOVZL 0,0
00435   143006          ADD 2,0
00436   000767          JMP LP+1
00437   034751 DN:      LDA 3,F
00440   175404          INC 3,3,SNR     ; FIRST OR SEC. ADDR?
00441   000405          JMP DN-1                ; SEC. ADDR. GO TO DN
00442   054746          STA 3,F         ; SAVE FIRST ADDR
00443   040742          STA 0,SAV
00444   102400          SUB 0,0
00445   000760          JMP LP+1                ; GET SEC. ADDR.
00446   101400          INC 0,0
00447   024732 DN:      LDA 1,CR
00450   054737          STA 3,SAV+2
00451   004470          JSR PRT
00452   024726          LDA 1,LF
00453   004466          JSR PRT
00454   034733          LDA 3,SAV+2
00455   030734          LDA 2,X
00456   050735          STA 2,SV        ; SV=SHIFT LOOP COUNTER
00457   152400          SUB 2,2
00460   050730          STA 2,F         ; F=PRINTED CPS ON LINE
00461   034724 RET:     LDA 3,SAV       ; SAV=BEG OR CURR ADDR
00462   151005          MOV 2,2,SNR     ; BEG OF LINE?
00463   000411          JMP LOP         ; SKP IF YES
00464   025400          LDA 1, ,3       ; LOAD CHARACTERS
00465   116432          SG 0,3
00466   000464          JMP DON
00467   175400          INC 3,3         ; INC ADDR
```

```
00470 030721          LDA 2,X
00471 050722          STA 2,SV
00472 054713          STA 3,SAV        ; SAVE CURR ADDR
00473 135000          MOV 1,3
00474 175100 LOP:     MOVL 3,3         ; SHIFT CHARACTER
00475 175100          MOVL 3,3
00476 024705          LDA 1,CM         ; LOAD MASK
00477 167400          AND 3,1          ; SAVE BITS 13-15
00500 030712          LDA 2,SX         ; MAKE THE NUMBER
00501 147000          ADD 2,1
00502 030711          LDA 2,SV         ; LOAD CHAR COUNTER
00503 040703          STA 0,SAV+1      ; SAVE AC0
00504 020705          LDA 0,X          ; LOAD FIRST CHAR NO.
00505 112414          SUB# 0,2,SNR     ;  IF NOT FIRST CHARACTER
00506 000403          JMP .+3          ; SKIP AHEAD
00507 030705          LDA 2,A          ; GET RID OF BIT 13
00510 147400          AND 2,1
00511 171000          MOV 3,2
00512 004427          JSR PRT
00513 020673          LDA 0,SAV+1      ; RESTORE AC0
00514 155000          MOV 2,3
00515 030676          LDA 2,SV
00516 151405          INC 2,2,SNR      ; LAST CHAR OF GP?
00517 000404          JMP OLP          ; SKP IF YES
00520 175100          MOVL 3,3         ; CIRCULAR SHIFT 3 BITS
00521 175100          MOVL 3,3
00522 175100          MOVL 3,3
00523 050667          STA 2,SV
00524 000752          JMP LOP+2
00525 030663 OLP:     LDA 2,F
00526 024667          LDA 1,B
00527 151400          INC 2,2
00530 130432          SZ 1,2           ; LAST CHAR ON LINE?
00531 000716          JMP DN           ; YES GO TO DN
00532 050656          STA 2,F
00533 024663          LDA 1,BLK        ; PRINT TWO BLANKS
00534 004405          JSR PRT
00535 004404          JSR PRT
00536 024653          LDA 1,X
00537 044654          STA 1,SV
00540 000721          JMP RET
00541 063511 PRT:     SKPBZ TTO        ; WAIT UNTIL PRINTER FREE
00542 000777          JMP .-1
00543 065111          DOAS 1,TTO       ; OUTPUT AC1
00544 001400          JMP 0,3          ; RETURN WHERE CALLED
00545 063610 RED:     SKPDN TTI
00546 000777          JMP .-1
00547 071510          DIAS 2,TTI
00550 071111          DOAS 2,TTO
00551 001400          JMP 0,3          ; RETURN
00552 024630 DON:     LDA 1,STR
00553 004766          JSR PRT
00554 024643          LDA 1,DEL
00555 004764          JSR PRT
00556 004763          JSR PRT
00557 004762          JSR PRT
00560 152000          SUB 2,2
00561 151400          INC 2,2
00562 150400          NEG 2,2
00563 050625          STA 2,F
```

```
---
   00564 024615          LDA 1,CR
   00565 004754          JSR PRT
   00566 024612          LDA 1,LF
   00567 004752          JSR PRT
   00570 000630          JMP START
   00571 063077          HALT
         000420          .END START


---

MODE:S400,571;
000400  000012  000015  000044  000047  000017  000405  000572  000001
000410  000001  177772  000060  177772  000061  000011  000040  000007
000420  024762  004520  062110  102400  024771  004520  034756  173400
000430  132432  000406  101120  101120  101120  143000  000767  034751
000440  173400  000405  054740  040742  102400  000760  101400  024732
000450  054737  004470  024726  004466  034733  030734  050735  152400
000460  050730  034724  151005  000411  025400  116432  000464  175400
000470  030721  050722  054713  135000  175100  175100  024705  167400
000500  030717  147700  030711  040703  020705  112414  000403  030705
000510  147400  171000  004427  020473  155000  030676  151405  000476
000520  175100  175100  175100  050670  000752  030663  024667  151400
000530  132432  000716  050656  024663  004405  004404  024653  044654
000540  000721  063511  000777  005111  001400  063610  000777  070510
000550  071111  001400  024632  004766  024643  004764  004763  004762
000560  152400  151400  152400  050625  024615  004754  024612  004752
000570  000630  063077  $
```

```
                        ECHO CHECK
CORE_SIZE=      4095    LOAD_POINT=       256    START_POINT=       272   RESTART_ADDRESS=       31;
NOVA(256)='0000000000001010'B;
NOVA(257)='0000000000001101'B;
NOVA(258)='0000000000010100'B;
NOVA(259)='0000000000010111'B;
NOVA(260)='0000000000011111'B;
NOVA(261)='0000000000000000'B;
NOVA(262)='0000000000000000'B;
NOVA(263)='0000000000000000'B;
NOVA(264)='1111111111111111'B;
NOVA(265)='1111111111111010'B;
NOVA(266)='0000000000110000'B;
NOVA(267)='0000000000000000'B;
NOVA(268)='0000000000110001'B;
NOVA(269)='0000000000001101'B;
NOVA(270)='0000000000100000'B;
NOVA(271)='0000000000000111'B;
NOVA(272)='0010100111110010'B;
NOVA(273)='0000100110101000'B;
NOVA(274)='0110000001001000'B;
NOVA(275)='1000010100000000'B;
NOVA(276)='0010100111111001'B;
NOVA(277)='0000100101010000'B;
NOVA(278)='0011100111101110'B;
NOVA(279)='1110111000000001'B;
NOVA(280)='1011010100011010'B;
NOVA(281)='0000000100000110'B;
NOVA(282)='1000010010100000'B;
NOVA(283)='1000010010100000'B;
NOVA(284)='1000000100101000'B;
NOVA(285)='1100011000000000'B;
NOVA(286)='0000000111110111'B;
NOVA(287)='0011100111101001'B;
NOVA(288)='1111101100000100'B;
NOVA(289)='0000000100000101'B;
NOVA(290)='0111011110000110'B;
NOVA(291)='0100001111100010'B;
NOVA(292)='1000010100000000'B;
NOVA(293)='0000001111110000'B;
NOVA(294)='1000011100000000'B;
NOVA(295)='0010100111101010'B;
NOVA(296)='0111011110111111'B;
NOVA(297)='0000100100111000'B;
NOVA(298)='0011010011101110'B;
NOVA(299)='0000100100111110'B;
NOVA(300)='0011100111011011'B;
NOVA(301)='0011000111011000'B;
NOVA(302)='0110100111011101'B;
NOVA(303)='1110101010000000'B;
NOVA(304)='0100100111101000'B;
NOVA(305)='0011100111101100'B;
NOVA(306)='1110100100001101'B;
NOVA(307)='0000000100001001'B;
NOVA(308)='0010101100000000'B;
NOVA(309)='1001110100111010'B;
NOVA(310)='0000000100111100'B;
NOVA(311)='1111101100000000'B;
NOVA(312)='0011000111101001'B;
```

```
NOVA(313)='0101003111G10010'B;
NOVA(314)='01011001110010110'B;
NOVA(315)='10111010000000000'B;
NOVA(316)='11111010010000000'B;
NOVA(317)='11111010010000000'B;
NOVA(318)='0010100111000101'B;
NOVA(319)='1110111100000000'B;
NOVA(320)='0011000111001010'B;
NOVA(321)='1100111000000000'B;
NOVA(322)='0011000111001001'B;
NOVA(323)='0100001110000011'B;
NOVA(324)='0010000111000101'B;
NOVA(325)='1000100100001100'B;
NOVA(326)='0000000100000011'B;
NOVA(327)='0011000111000101'B;
NOVA(328)='1100111100000000'B;
NOVA(329)='1111001000000000'B;
NOVA(330)='0000010100001111'B;
NOVA(331)='0010000110111011'B;
NOVA(332)='1100110100000000'B;
NOVA(333)='0010000110111110'B;
NOVA(334)='1101001100000101'B;
NOVA(335)='0000000100000110'B;
NOVA(336)='1111101001000000'B;
NOVA(337)='1111101001000000'B;
NOVA(338)='1111101001000000'B;
NOVA(339)='0100001110111000'B;
NOVA(340)='0000000111101010'B;
NOVA(341)='0011000111011011'B;
NOVA(342)='0010010011011111'B;
NOVA(343)='1100100110000000'B;
NOVA(344)='1011010100011010'B;
NOVA(345)='0000000111000111'B;
NOVA(346)='0100000110101110'B;
NOVA(347)='0010100110111011'B;
NOVA(348)='0000100100000101'B;
NOVA(349)='0000100100000100'B;
NOVA(350)='0010100111010111'B;
NOVA(351)='0100100111010100'B;
NOVA(352)='0000000111010001'B;
NOVA(353)='0110011101001001'B;
NOVA(354)='0000000111111111'B;
NOVA(355)='0110100100100001'B;
NOVA(356)='0000000110000000'B;
NOVA(357)='0110011110001000'B;
NOVA(358)='0000000111111111'B;
NOVA(359)='0111000101001000'B;
NOVA(360)='0111001001001001'B;
NOVA(361)='0000001100000000'B;
NOVA(362)='0010100110011000'B;
NOVA(363)='0000100111111110'B;
NOVA(364)='0010100110101011'B;
NOVA(365)='0000100111110100'B;
NOVA(366)='0000100111111011'B;
NOVA(367)='0000100111111010'B;
NOVA(368)='1100100100000000'B;
NOVA(369)='1100100110000000'B;
NOVA(370)='1100100100000000'B;
NOVA(371)='0100100110001101'B;
NOVA(372)='0010100110001101'B;
```

```
NOVA(373)='0J0010J111101100'B;
NOVA(374)='00101J0110001010'B;
NOVA(375)='0000100111101010'B;
NOVA(376)='00J0000J10011000'B;
NOVA(377)='0110011000111111'B;
NOVA(378)='0000000010010000'B;

                    END OF ECHO CHECK
```

```
'00000000000110000'B
'00000000000110001'B
'00000000000110010'B
'00000000000110011'B
'00000000000110100'B
'00000000000110101'B
'00000000000110110'B
'00000000000110111'B
'00000000000111000'B
'00000000000111001'B
'00000000000111010'B
'00000000000111011'B
'00000000000111100'B
'00000000000111101'B
'00000000000111110'B
'00000000000111111'B
'00000000001000000'B
'00000000001000001'B
'00000000001000010'B
'00000000001000011'B
'00000000001000100'B
'00000000001000101'B
'00000000001000110'B
'00000000001000111'B
'00000000001001000'B
'00000000001001001'B
'00000000001001010'B
'00000000001001011'B
'00000000001001100'B
'00000000001001101'B
'00000000001001110'B
'00000000001001111'B
'00000000001010000'B
'00000000001010001'B
'00000000001010010'B
'00000000001010011'B
'00000000001010100'B
'00000000001010101'B
'00000000001010110'B
'00000000001010111'B
'00000000001011000'B
'00000000001011001'B
'00000000001011010'B
'00000000001011011'B
'00000000001011100'B
'00000000001011101'B
'00000000001011110'B
'00000000001011111'B
'00000000001100000'B
'00000000001100001'B
'00000000001100010'B
'00000000001100011'B
'00000000001100100'B
'00000000001100101'B
'00000000001100110'B
'00000000001100111'B
'00000000001101000'B
'00000000001101001'B
'00000000001101010'B
'00000000001101011'B
'00000000001101100'B
'00000000001101101'B
'00000000001101110'B
'00000000001101111'B
'00000000001110000'B
'00000000001110001'B
'00000000001110010'B
'00000000001110011'B
'00000000001110100'B
'00000000001110101'B
'00000000001110110'B
'00000000001110111'B
'00000000001111000'B
'00000000001111001'B
'00000000001111010'B
'00000000001111011'B
'00000000001111100'B
'00000000001111101'B
'00000000001111110'B
'00000000001111111'B
'00000000010000000'B
'00000000010000001'B
'00000000010000010'B
'00000000010000011'B
'00000000010000100'B
'00000000010000101'B
'00000000010000110'B
'00000000010000111'B
'00000000010001000'B
'00000000010001001'B
'00000000010001010'B
'00000000010001011'B
'00000000010001100'B
'00000000010001101'B
'00000000010001110'B
'00000000010001111'B
'00000000010010000'B
'00000000010010001'B
'00000000010010010'B
'00000000010010011'B
'00000000010010100'B
'00000000010010101'B
'00000000010010110'B
'00000000010010111'B
'00000000010011000'B
'00000000010011001'B
'00000000010011010'B
'00000000010011011'B
'00000000010011100'B
'00000000010011101'B
'00000000010011110'B
'00000000010011111'B
'00000000010100000'B
'00000000010100001'B
'00000000010100010'B
'00000000010100011'B
'00000000010100100'B
'00000000010100101'B
'00000000010100110'B
'00000000010100111'B
'00000000010101000'B
'00000000010101001'B
'00000000010101010'B
'00000000010101011'B
'00000000010101100'B
'00000000010101101'B
'00000000010101110'B
'00000000010101111'B
'00000000010110000'B
'00000000010110001'B
'00000000010110010'B
'00000000010110011'B
'00000000010110100'B
'00000000010110101'B
'00000000010110110'B
'00000000010110111'B
'00000000010111000'B
'00000000010111001'B
'00000000010111010'B
'00000000010111011'B
'00000000010111100'B
'00000000010111101'B
'00000000010111110'B
'00000000010111111'B
'00000000011000000'B
'00000000011000001'B
'00000000011000010'B
'00000000011000011'B
'00000000011000100'B
'00000000011000101'B
'00000000011000110'B
'00000000011000111'B
'00000000011001000'B
'00000000011001001'B
'00000000011001010'B
'00000000011001011'B
'00000000011001100'B
'00000000011001101'B
'00000000011001110'B
'00000000011001111'B
'00000000011010000'B
'00000000011010001'B
'00000000011010010'B
'00000000011010011'B
'00000000011010100'B
'00000000011010101'B
'00000000011010110'B
'00000000011010111'B
'00000000011011000'B
'00000000011011001'B
'00000000011011010'B
'00000000011011011'B
'00000000011011100'B
'00000000011011101'B
'00000000011011110'B
'00000000011011111'B
'00000000011100000'B
'00000000011100001'B
'00000000011100010'B
'00000000011100011'B
'00000000011100100'B
'00000000011100101'B
'00000000011100110'B
'00000000011100111'B
'00000000011101000'B
'00000000011101001'B
'00000000011101010'B
'00000000011101011'B
'00000000011101100'B
'00000000011101101'B
'00000000011101110'B
'00000000011101111'B
'00000000011110000'B
'00000000011110001'B
'00000000011110010'B
'00000000011110011'B
'00000000011110100'B
'00000000011110101'B
'00000000011110110'B
'00000000011110111'B
'00000000011111000'B
'00000000011111001'B
'00000000011111010'B
'00000000011111011'B
'00000000011111100'B
'00000000011111101'B
'00000000011111110'B
'00000000011111111'B
'00000000100000000'B
'00000000100000001'B
'00000000100000010'B
'00000000100000011'B
'00000000100000100'B
'00000000100000101'B
'00000000100000110'B
'00000000100000111'B
'00000000100001000'B
'00000000100001001'B
'00000000100001010'B
'00000000100001011'B
'00000000100001100'B
'00000000100001101'B
'00000000100001110'B
'00000000100001111'B
'00000000100010000'B
'00000000100010001'B
'00000000100010010'B
'00000000100010011'B
'00000000100010100'B
'00000000100010101'B
'00000000100010110'B
'00000000100010111'B
'00000000100011000'B
'00000000100011001'B
'00000000100011010'B
'00000000100011011'B
'00000000100011100'B
'00000000100011101'B
'00000000100011110'B
'00000000100011111'B
'00000000100100000'B
'00000000100100001'B
'00000000100100010'B
'00000000100100011'B
'00000000100100100'B
'00000000100100101'B
'00000000100100110'B
'00000000100100111'B
'00000000100101000'B
'00000000100101001'B
'00000000100101010'B
'00000000100101011'B
'00000000100101100'B
'00000000100101101'B
'00000000100101110'B
'00000000100101111'B
'00000000100110000'B
'00000000100110001'B
'00000000100110010'B
'00000000100110011'B
'00000000100110100'B
'00000000100110101'B
'00000000100110110'B
'00000000100110111'B
'00000000100111000'B
'00000000100111001'B
'00000000100111010'B
'00000000100111011'B
'00000000100111100'B
'00000000100111101'B
'00000000100111110'B
'00000000100111111'B
'00000000101000000'B
'00000000101000001'B
'00000000101000010'B
'00000000101000011'B
'00000000101000100'B
'00000000101000101'B
'00000000101000110'B
'00000000101000111'B
'00000000101001000'B
'00000000101001001'B
'00000000101001010'B
'00000000101001011'B
'00000000101001100'B
'00000000101001101'B
'00000000101001110'B
'00000000101001111'B
'00000000101010000'B
'00000000101010001'B
'00000000101010010'B
'00000000101010011'B
'00000000101010100'B
'00000000101010101'B
'00000000101010110'B
'00000000101010111'B
'00000000101011000'B
'00000000101011001'B
'00000000101011010'B
'00000000101011011'B
'00000000101011100'B
'00000000101011101'B
'00000000101011110'B
'00000000101011111'B
'00000000101100000'B
'00000000101100001'B
'00000000101100010'B
'00000000101100011'B
'00000000101100100'B
'00000000101100101'B
'00000000101100110'B
'00000000101100111'B
'00000000101101000'B
'00000000101101001'B
'00000000101101010'B
'00000000101101011'B
'00000000101101100'B
'00000000101101101'B
'00000000101101110'B
'00000000101101111'B
'00000000101110000'B
'00000000101110001'B
'00000000101110010'B
'00000000101110011'B
'00000000101110100'B
'00000000101110101'B
'00000000101110110'B
'00000000101110111'B
'00000000101111000'B
'00000000101111001'B
'00000000101111010'B
'00000000101111011'B
'00000000101111100'B
'00000000101111101'B
'00000000101111110'B
'00000000101111111'B
```

```
'0000000000110000'B   '0000000000100000'B   '0000000000100000'B   '0000000000110001'B   '0000000000110111'B
'0000000000110101'B   '0000000000110001'B   '0000000000110000'B   '0000000000110000'B   '0000000000100000'B
'0000000000100000'B   '0000000000110000'B   '0000000000110010'B   '0000000000110100'B   '0000000000110111'B
'0000000000110000'B   '0000000000110101'B   '0000000000100000'B   '0000000000100000'B   '0000000000110001'B
'0000000000110110'B   '0000000000110111'B   '0000000000110100'B   '0000000000110000'B   '0000000000110000'B
'0000000000001101'B   '0000000000001010'B   '0000000000110000'B   '0000000000110000'B   '0000000000110000'B
'0000000000110101'B   '0000000000110000'B   '0000000000110000'B   '0000000000100000'B   '0000000000100000'B
'0000000000110000'B   '0000000000110011'B   '0000000000110000'B   '0000000000110111'B   '0000000000110001'B
'0000000000110010'B   '0000000000100000'B   '0000000000100000'B   '0000000000110001'B   '0000000000110100'B
'0000000000110111'B   '0000000000110000'B   '0000000000110000'B   '0000000000110000'B   '0000000000100000'B
'0000000000100000'B   '0000000000110000'B   '0000000000110011'B   '0000000000110000'B   '0000000000110111'B
'0000000000110001'B   '0000000000110001'B   '0000000000100000'B   '0000000000100000'B   '0000000000110000'B
'0000000000110100'B   '0000000000110000'B   '0000000000110111'B   '0000000000110000'B   '0000000000110011'B
'0000000000100000'B   '0000000000100000'B   '0000000000110000'B   '0000000000110010'B   '0000000000110000'B
'0000000000110111'B   '0000000000110000'B   '0000000000110101'B   '0000000000100000'B   '0000000000100000'B
'0000000000110001'B   '0000000000110001'B   '0000000000110010'B   '0000000000110100'B   '0000000000110001'B
'0000000000110100'B   '0000000000100000'B   '0000000000100000'B   '0000000000110000'B   '0000000000110000'B
'0000000000110000'B   '0000000000110100'B   '0000000000110000'B   '0000000000110011'B   '0000000000100000'B
'0000000000100000'B   '0000000000110000'B   '0000000000110011'B   '0000000000110000'B   '0000000000110111'B
'0000000000110000'B   '0000000000110101'B   '0000000000001101'B   '0000000000001010'B   '0000000000110000'B
'0000000000110000'B   '0000000000110000'B   '0000000000110101'B   '0000000000110001'B   '0000000000110000'B
'0000000000100000'B   '0000000000100000'B   '0000000000110001'B   '0000000000110100'B   '0000000000110111'B
'0000000000110100'B   '0000000000110000'B   '0000000000110000'B   '0000000000100000'B   '0000000000100000'B
'0000000000110001'B   '0000000000110111'B   '0000000000110001'B   '0000000000110000'B   '0000000000110000'B
'0000000000110000'B   '0000000000100000'B   '0000000000100000'B   '0000000000110000'B   '0000000000110000'B
'0000000000110100'B   '0000000000110100'B   '0000000000110010'B   '0000000000110111'B   '0000000000100000'B
'0000000000100000'B   '0000000000110000'B   '0000000000110010'B   '0000000000110000'B   '0000000000110110'B
'0000000000110111'B   '0000000000110011'B   '0000000000100000'B   '0000000000100000'B   '0000000000110001'B
'0000000000110101'B   '0000000000110101'B   '0000000000110000'B   '0000000000110000'B   '0000000000110000'B
'0000000000100000'B   '0000000000100000'B   '0000000000110000'B   '0000000000110011'B   '0000000000110000'B
'0000000000110110'B   '0000000000110111'B   '0000000000110110'B   '0000000000100000'B   '0000000000100000'B
'0000000000110001'B   '0000000000110101'B   '0000000000110001'B   '0000000000110100'B   '0000000000110000'B
'0000000000110101'B   '0000000000100000'B   '0000000000100000'B   '0000000000110000'B   '0000000000110000'B
'0000000000110000'B   '0000000000110100'B   '0000000000110000'B   '0000000000110110'B   '0000000000001101'B
'0000000000001010'B   '0000000000110000'B   '0000000000110000'B   '0000000000110000'B   '0000000000110101'B
'0000000000110011'B   '0000000000110000'B   '0000000000100000'B   '0000000000100000'B   '0000000000100100'B
'0000000000000111'B   '0000000000000111'B   '0000000000000111'B   '0000000000001101'B   '0000000000001010'B
'0000000000100100'B

EOF ENCOUNTERED ON NOVA.SYSIN. JOE HALTED

PC=     359;

REG(0)='0000000000010001'B                REG(1)='0000000000001001'B                REG(2)='1111111111111111
'B                REG(3)='0000000100010110'B;

SHIFTER='0000000000000000'B                CARRY='0'B;
NOVA(256)='0000000000001010'B;                NOVA(257)='0000000000001101'B;                NOVA(258)='0000000000100
100'B;                NOVA(259)='0000000000000111'B;                NOVA(260)='0000000000001111'B;
NOVA(261)='0000000101010000'B;                NOVA(262)='0000000101010000'B;                NOVA(263)='0000000100000
110'B;                NOVA(264)='1111111111111111'B;                NOVA(265)='1111111111111010'B;
NOVA(266)='0000000000110000'B;                NOVA(267)='1111111111111010'B;                NOVA(268)='0000000000110
001'B;                NOVA(269)='0000000000001001'B;                NOVA(270)='0000000000100000'B;
NOVA(271)='0000000000000111'B;                NOVA(272)='0010100111110010'B;                NOVA(273)='0000100101010
000'B;                NOVA(274)='0110000001001000'B;                NOVA(275)='1000010100000000'B;
NOVA(276)='0010100111111001'B;                NOVA(277)='0000100101010000'B;                NOVA(278)='0011100111101
110'B;                NOVA(279)='1111011100000000'B;                NOVA(280)='1011010100011010'B;
NOVA(281)='0000000100000110'B;                NOVA(282)='1000001001010000'B;                NOVA(283)='1000001001010
000'B;                NOVA(284)='1000001001010000'B;                NOVA(285)='1100011000000000'B;
```

```
NOVA(286)='0000000111110111'B;  NOVA(287)='0011100111101001'B;  NOVA(288)='1111101100000
100'B;  NOVA(289)='0000000100000101'B;  NOVA(290)='0101100111100110'B;
NOVA(291)='0100000111100010'B;  NOVA(292)='1000010100000000'B;  NOVA(293)='0000000111110
000'B;  NOVA(294)='1000001100000000'B;  NOVA(295)='0010100111011010'B;
NOVA(296)='0101100111011111'B;  NOVA(297)='0000100100111000'B;  NOVA(298)='0010100111010
110'B;  NOVA(299)='0000100100110110'B;  NOVA(300)='0011100111011011'B;
NOVA(301)='0011000111011100'B;  NOVA(302)='0101000111011101'B;  NOVA(303)='1101010100000
000'B;  NOVA(304)='0101000111011000'B;  NOVA(305)='0011100111010100'B;
NOVA(306)='1101010100000101'B;  NOVA(307)='0000000100001001'B;  NOVA(308)='0010101100000
000'B;  NOVA(309)='1001110100011010'B;  NOVA(310)='0000000100110100'B;
NOVA(311)='1111101100000000'B;  NOVA(312)='0011000111010001'B;  NOVA(313)='0101000111010
010'B;  NOVA(314)='0101100111001011'B;  NOVA(315)='1011101000000000'B;
NOVA(316)='1111101001000000'B;  NOVA(317)='1111101001000000'B;  NOVA(318)='0010100111000
101'B;  NOVA(319)='1110111100000000'B;  NOVA(320)='0011000111001010'B;
NOVA(321)='1101111000000000'B;  NOVA(322)='0011000111001001'B;  NOVA(323)='0100000111000
011'B;  NOVA(324)='0010000111000101'B;  NOVA(325)='1001010100001100'B;
NOVA(326)='0000000100000011'B;  NOVA(327)='0011000111000101'B;  NOVA(328)='1100111100000
000'B;  NOVA(329)='1111001000000000'B;  NOVA(330)='0000010100010111'B;
NOVA(331)='0010000110111011'B;  NOVA(332)='1101101000000000'B;  NOVA(333)='0011000110111
110'B;  NOVA(334)='1101001100000101'B;  NOVA(335)='0000000100000110'B;
NOVA(336)='1111110001000000'B;  NOVA(337)='1111101001000000'B;  NOVA(338)='1111101001000
000'B;  NOVA(339)='0101000110111000'B;  NOVA(340)='0000000111101010'B;
NOVA(341)='0011000110111011'B;  NOVA(342)='0010100110110111'B;  NOVA(343)='1101001100000
000'B;  NOVA(344)='1011010100110100'B;  NOVA(345)='0000000111001110'B;
NOVA(346)='0101000110101110'B;  NOVA(347)='0010100110110011'B;  NOVA(348)='0000100100000
101'B;  NOVA(349)='0000100100000100'B;  NOVA(350)='0010100110101011'B;
NOVA(351)='0100100110101100'B;  NOVA(352)='0000000111010001'B;  NOVA(353)='0110011101001
001'B;  NOVA(354)='0100000111111111'B;  NOVA(355)='0110101001001001'B;
NOVA(356)='0000000110000000'B;  NOVA(357)='0110011110001000'B;  NOVA(358)='0000000111111
111'B;  NOVA(359)='0111000101000000'B;  NOVA(360)='0111001001001001'B;
NOVA(361)='0000000110000000'B;  NOVA(362)='0010100110011000'B;  NOVA(363)='0000100111111
110'B;  NOVA(364)='0010100110100011'B;  NOVA(365)='0000100111110100'B;
NOVA(366)='0000100111111011'B;  NOVA(367)='0000100111110010'B;  NOVA(368)='1101010100000
000'B;  NOVA(369)='1101001100000000'B;  NOVA(370)='1101000100000000'B;
NOVA(371)='0100000110001101'B;  NOVA(372)='0010100110001101'B;  NOVA(373)='0000100111101
100'B;  NOVA(374)='0100100110001000'B;  NOVA(375)='0000100111101010'B;
NOVA(376)='0000000100000000'B;  NOVA(377)='0110011000111111'B;  NOVA(378)='0000000100000
000'B;
```

A SOFTWARE SIMULATOR TO HOST THE DATA GENERAL NOVA ON THE IBM 360

by

SUNEE GADETRAGOON

B.Sc.(Hons.), Chulalongkorn University, Bangkok
1964

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1972

In order to use programs originally written for the Data General Nova on the IBM 360, a software simulator has been designed. This simulator consists of a single program written in IBM System 360 PL/1.

The input to the simulator is the Nova Object Code, written as 16 digit binary number. The simulator, when invoked, behaves as the central processor of the Data General Nova. It reads the object code of the Data General Nova into a variable size simulated memory and then starts simulating the Nova central processor by interpreting and executing the Nova Object Code in terms of the available 360 facilities.

The simulator is capable of simulating the Nova central processor for all the different instructions of the Nova, except the Input/Output instructions.

The report consists of a brief description of the Data General Nova, a detailed description of the working of the simulator, instructions on how to use the simulator, a discussion of the testing and debugging scheme of the simulator and a brief discussion on making the simulator more efficient.