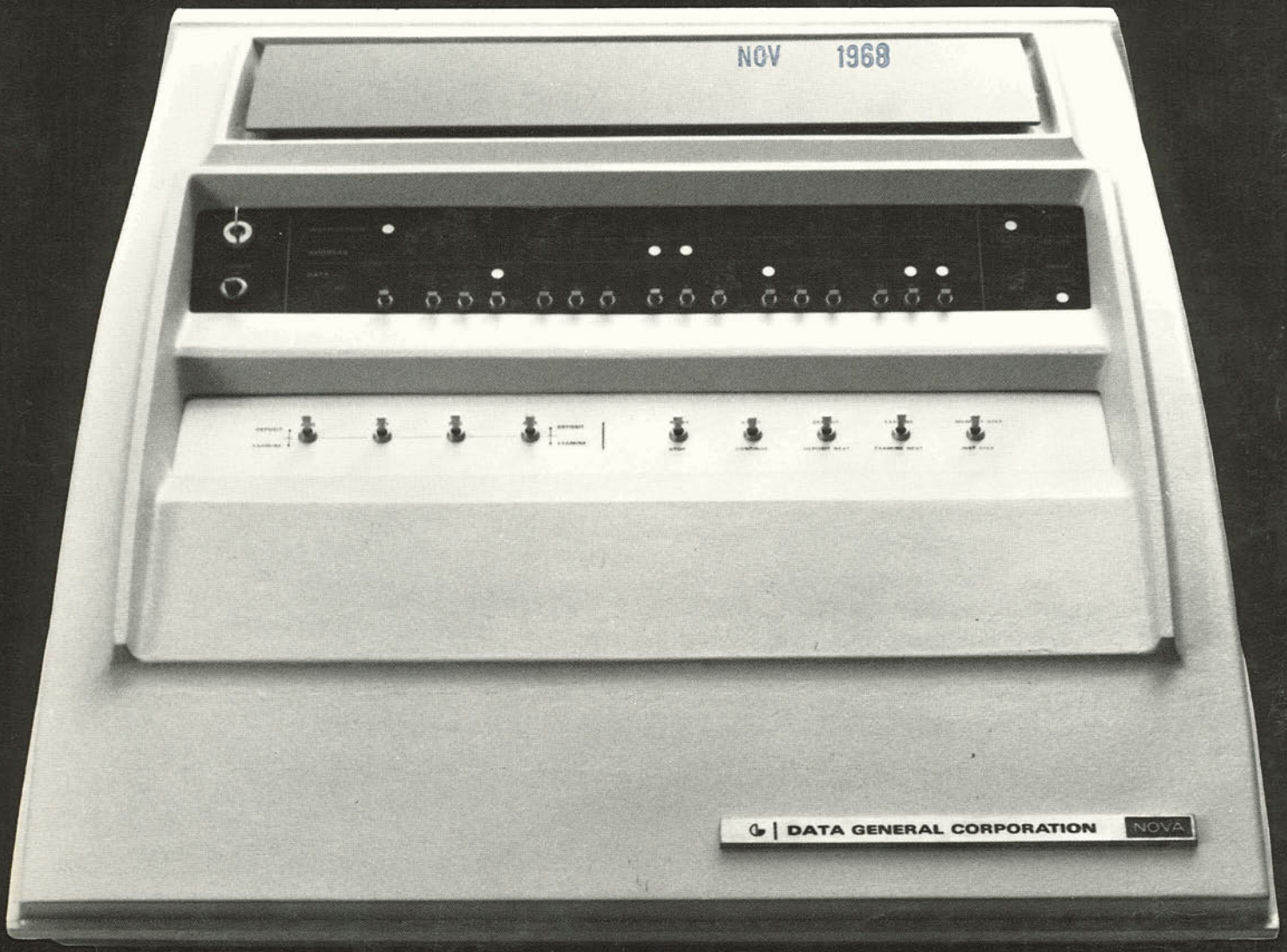
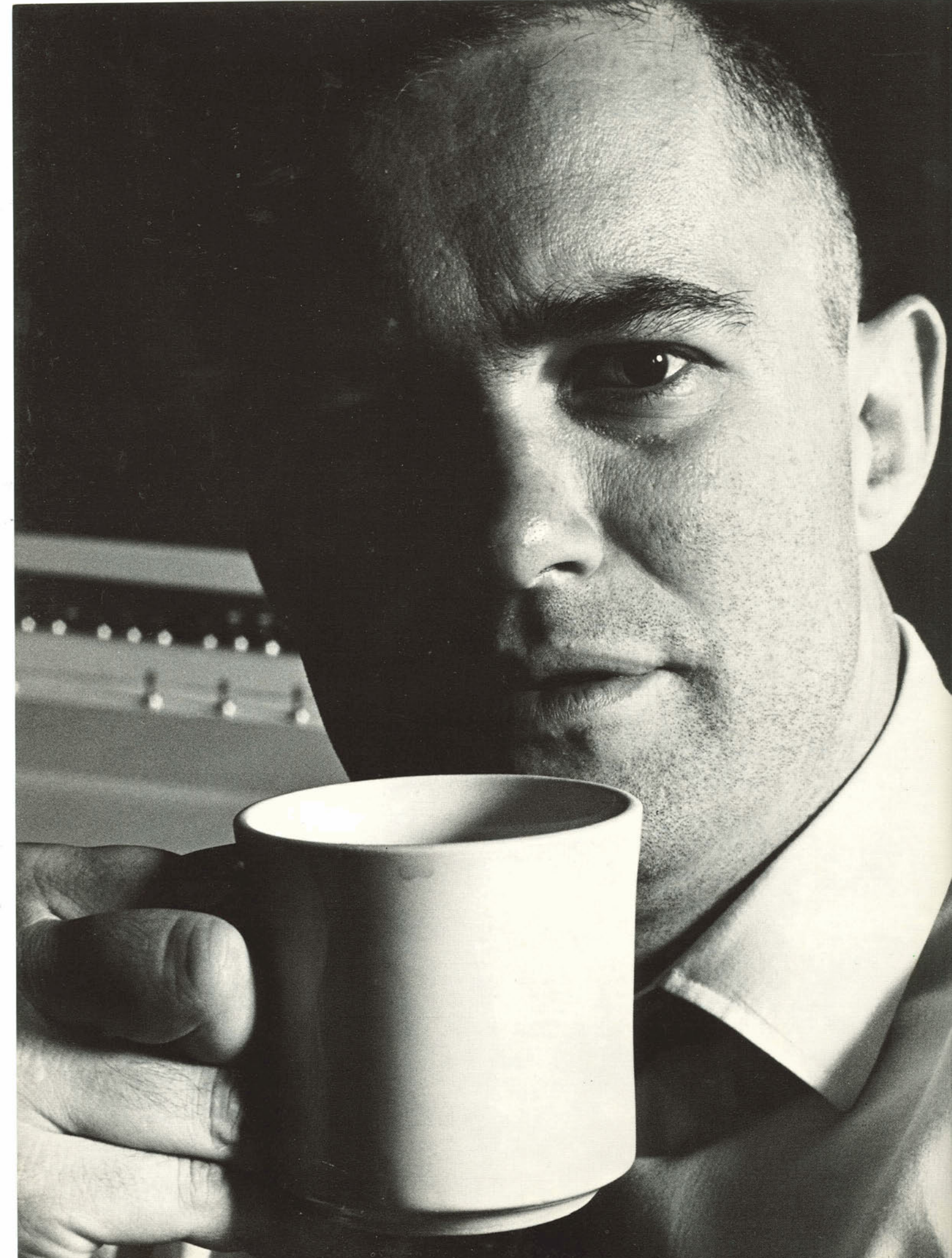
The best small computer in the world.





Edson D. de Castro, President, Data General Corporation

A hard look at small computers

Small computers are creating quite a stir.

You hear that they are being purchased faster than any other kind of computer. And that they are creating an applications explosion, expanding even their current rate of sales growth.

It seems manufacturers have not let this action go unnoticed. The expanding marketplace is encouraging a rush of new products. Old designs are being revamped. New designs are being rushed to market. The mini-computer boom is on.

But what is the boom producing? Is a growing market and increasing competition producing better computer value?

Well, the current crop of machines varies considerably. Word lengths vary. Input/output facilities vary. Instruction sets vary. Software varies. Performance varies.

We designed the NOVA believing that new technology and improvements in the art of computer design had offered the opportunity for vast improvements in the price/performance ratio of the minimum configuration of small general purpose digital computers.

But there were some things we had to know before we could take advantage of these opportunities.

We had to know that, mini-computer or not, the power of these machines stems from their performance as general purpose computers. That whatever they are used for, they would have to be programmed first. And that it was possible to so strip a computer of programming power that the machine could become more trouble than it was worth.

At the same time, we had to know that the special domain of these machines is in the on-line, real-time environment. That over half the small computers purchased are built into larger systems as special purpose data reducers and controllers.

I think you will find the NOVA to be a better small computer. We designed the system incorporating an architecture previously found only in medium and large scale third generation computers. This architecture permitted an extremely powerful instruction set. We took full advantage of medium scale integration. We were also able to design the NOVA around the special input/output and packaging requirements of the small computer. We designed a new kind of read-only memory interchangeable with core, for "black box" configurations.

You see, in designing and pricing the NOVA, we were after more than just getting a piece of the small computer action. We were after a machine that would be the basis for starting a major computer company. I think we succeeded. The NOVA is that good.

The hardware

As Ed has mentioned, the architecture of the NOVA had previously never been implemented in a small computer design. The breakthrough that made it possible was medium scale integration.

It now is possible to obtain sixteen flip-flops in a single package for use in regular registers; the cost per function of an MSI component is somewhat lower than the cost of discrete gates and flip-flops. But that is not MSI's major contribution. The big savings with MSI came from the reduction of interconnection and resultant savings in packaging costs. And, of course, the reduction of interconnection also had a very favorable effect on reliability. MSI accounts for over half the gates and flip-flops used in the NOVA.

And there are two kinds of memory available with the NOVA, core and read-only.

A user will create his read-only memory by writing his program using a core memory. He uses the full software of the system to write, edit, and debug his program. After he is satisfied with his program, he dumps it on paper tape and sends the tape to Data General. This tape is used as the basis for manufacturing the memory and verifying its contents. When complete, the read-only memory is simply plugged into the NOVA in place of the core memory. Should changes be required in the read-only memory they can easily be made by a technician.

These two kinds of memory are homogenous; the alterable and read-only storage are treated identically by the program and the processor. The only difference is that different kinds of memory vary in speed. But the two kinds of memory may be mixed. In a system in which only one program is being used, it can be stored along with its constants, in a read-only memory module. The alterable core memory can then be used for the storage of variable data and intermediary results. Or, the system console can be removed and the system operated as a hard wired controller. By changing read-only memories, the functions performed by this controller may be altered. Memories are available in 1K, 2K, or 4K word modules. The maximum memory size is 32K words or 64K bytes.

The read-write cycle times for the several core memory modules and the access time for read-only memory are as follows:

1K	6.5	micro-seconds
2K	3.9	
4K	2.6	
Read-only Memory	2.4	

In and out

Small computers interface with more kinds of devices with greater varying data rates and priority interrupt requirements than any other class of computer. Not only do they interface with the full range of computer peripherals, but they often become part of special or unique systems.

We decided to build a complete input/output facility into the basic NOVA. Included in the system are facilities for program interrupt and high speed data transfers with provision for direct access to memory. Any device can interrupt the normal program flow on a priority basis. A high speed device such as magnetic tape or disk can gain direct access to memory through a data channel without requiring the execution of any instructions. The data channel logic allows the transfer of data to or from memory, incrementing of memory word, and adding external data to a word already in memory. The latter two features facilitate such functions as signal averaging and pulse height analysis.

The NOVA I/O facility consists of 16 bi-directional data lines, 6 lines for device selection, 19 unary control lines from the central processor, and 6 control lines to the central processor. The control lines from the central processor are used to synchronize all data transfers on the data lines, to initiate and stop device functions, and to control the priority interrupt system. The control lines to the processor are used to indicate device status, and to request priority interrupt and data channel service.

The unary control lines from the processor contain two types of information: the specific function to be performed by the device, and timing information. These control lines are arranged in such a way that the device need connect only to those that correspond to the particular I/O functions that the device requires. The timing of the control lines is determined by the processor in such a fashion that the device does not require any time-dependent circuits to connect to the I/O interface.

The input/output system allows the program to address up to 62 external devices.

Henry Burkhardt, III, Vice President, Data General Corporation

Instruction power

Most medium and large scale third generation computer systems have central processors organized around multiple general purpose registers or accumulators. The logical and arithmetic instructions of these machines are performed by manipulating the contents of these accumulators. There is less need to address or access memory. Also, the availablility of these multiple registers improves the efficiency of accumulator to memory operations and data flow between the computer and peripheral devices.

Until we designed the NOVA, small computers either had a single accumulator or assigned memory locations to simulate this organization, trading off much of the basic power of the set-up.

In the NOVA we have been able to fully implement a multi-accumulator central processor architecture. It is from this architecture that much of the power of the machine stems.

The NOVA has four full sixteen-bit word accumulators, two of which may be used as index registers. Data can be moved in either direction between any memory location and any NOVA accumulator. Although a word in memory can be incremented or decremented, all other arithmetic and logical operations are performed on operands in the accumulators, with the result appearing in an accumulator. Associated with the accumulators is a single carry flag, which indicates when the magnitude of the result is too large to be accommodated in a single accumulator. The left and right halves of any accumulator can be swapped, the contents of any accumulator can be tested for a skip, and the 17-bit word contained in any accumulator combined with the carry can be rotated right or left. An instruction that references memory can address two of the accumulators as index registers and transfers to and from peripheral devices are also made through the accumulators.

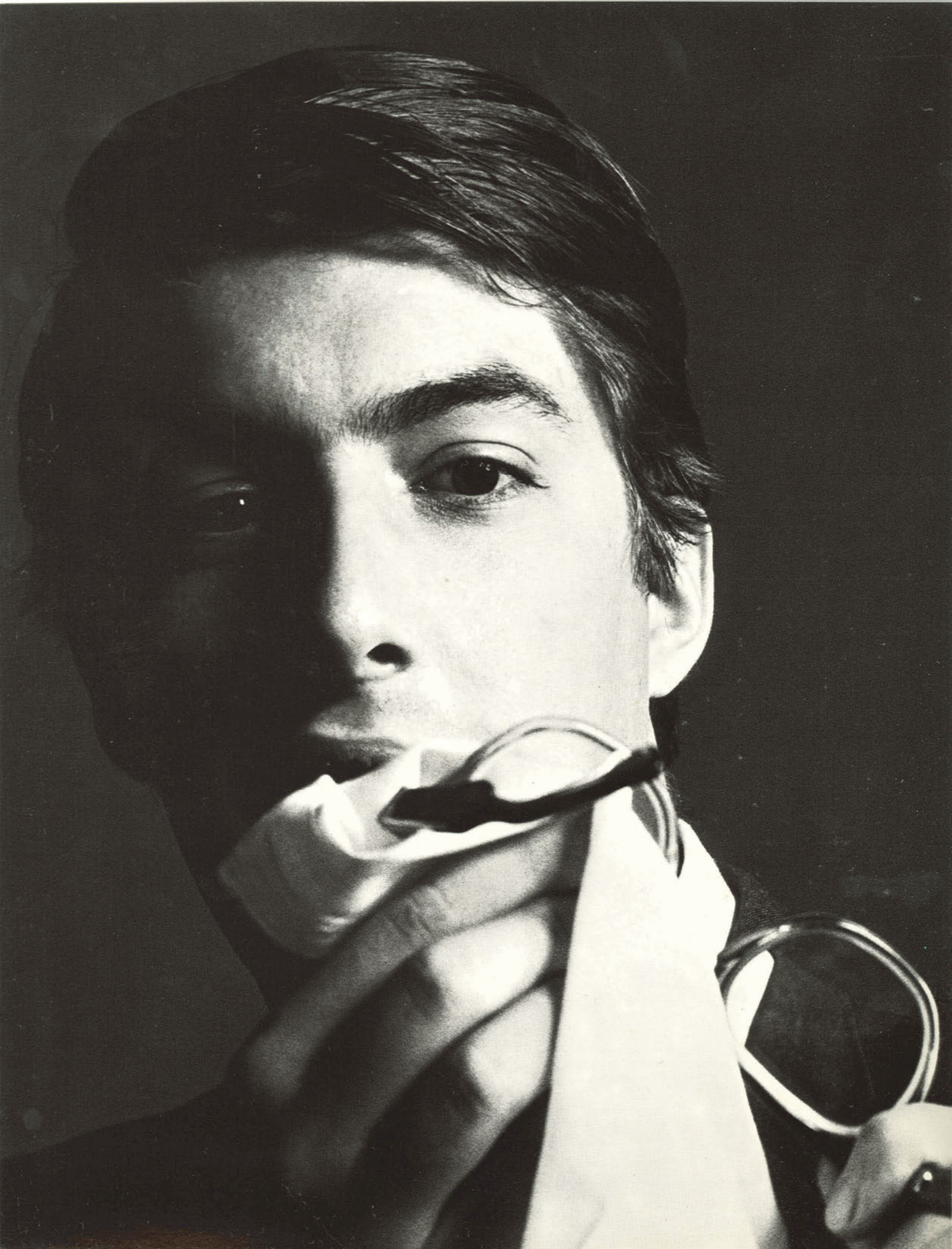
This multi-accumulator organization cuts down on the number of instructions necessary to execute a program. And reduces the amount of data movement in the machine. For example, in as trivial an operation as the exchanging of the contents of two memory locations, the multi-accumulator set-up reduces the number of instructions by one third.

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.

Arithmetic and logical instructions are frequently preceded by instructions which modify an operand and followed by a modification of the result and sometimes by a test. We felt that if these operations could be combined in a single instruction class, a much simpler to use and more powerful instruction structure would be achieved. We designed a class of instructions arranged so that each bit has its own function and thus it is unnecessary to decode most portions of the instruction word. The same instruction that adds or subtracts can also shift the result or swap its halves, test the result and /or carry for a skip, and specify whether or not the result shall actually be retained.

A single input /output instruction can transfer a word between an accumulator and a device and at the same time control the device operation.

The NOVA is much easier to program than single accumulator machines. The results of address calculations are immediately available for index purposes to the memory reference instructions. One accumulator can be used for in-out data transmission without disturbing others being used continually for computations. Complex software routines such as multiplication, division and floating point can be performed without constantly referencing memory.



Henry Burkhardt, III, Vice President, Data General Corporation

Software: first things first

Frankly, the basic NOVA software has been designed for the experienced computer user.

We knew that we would never stop developing new software packages and initially, we decided to concentrate on those things that were most integrally a part of the system.

The initial NOVA software includes a powerful assembler, a context oriented text editor, a multiple breakpoint debugger, complete hardware diagnostics, utility programs, and mathematical routines, including floating point arithmetic.

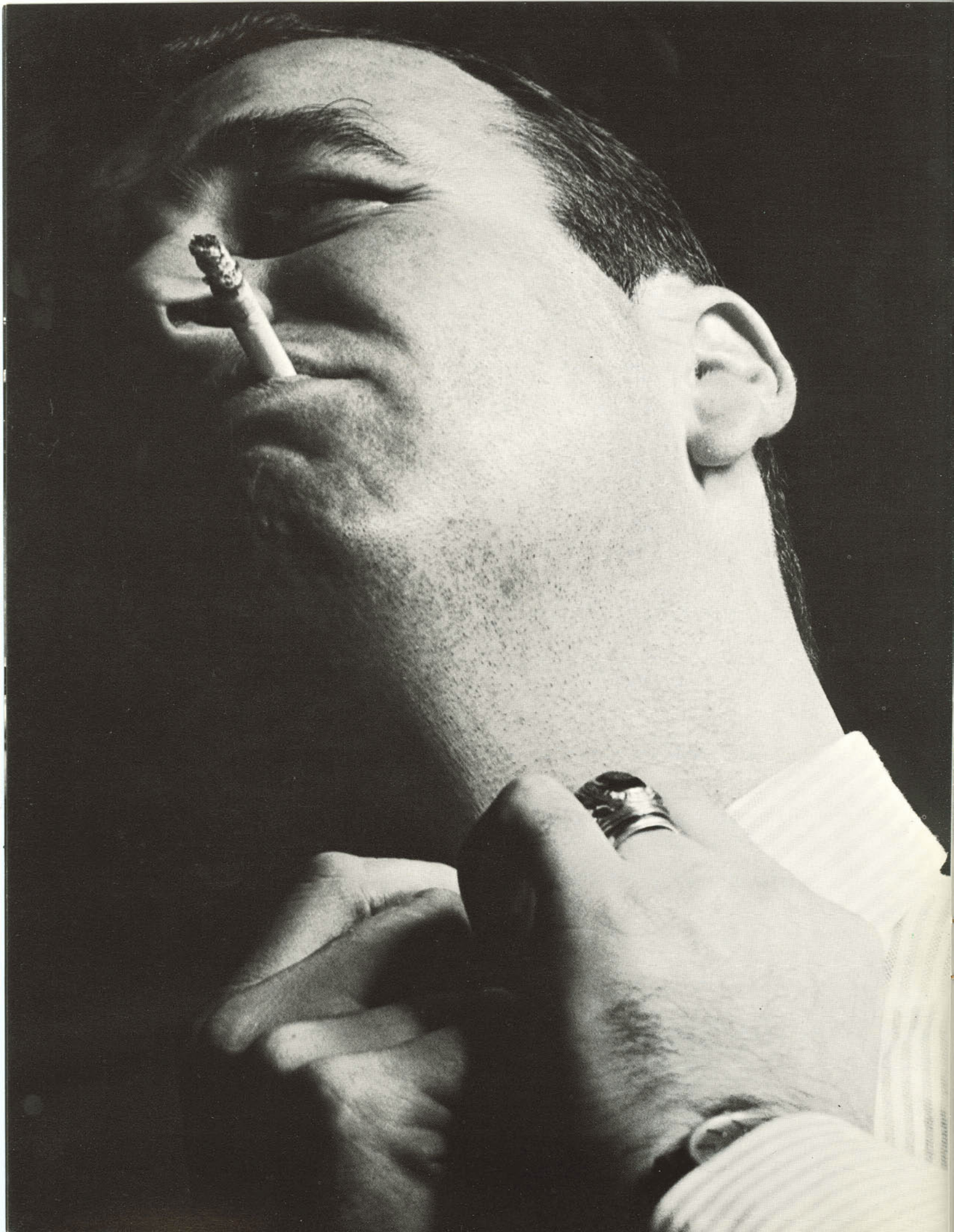
The NOVA assembler is a two pass system producing absolute binary and an assembly listing. Pseudo commands are provided to alter assembly origin, radix and to define new operation codes. Text may also be processed and packed into binary words. Input/output is fully buffered using the priority interrupt system, a binary search is used for the symbol table, and hence the assembly speed is I/O limited. The assembly language is free-form. The input need not be precisely formatted into columns as is required by many small computer assemblers. Control characters are used to delimit labels, comments and instruction fields. This provides greater freedom in the generation of program text as well as vastly reducing the errors due to missing spaces or blanks. The basic philosophy of the assembler has been to provide as few "default conditions" as possible. If it isn't entirely obvious what the user intended by a given line of code, the assembler will flag the line as questionable.

Since very few small computers are operated in an environment where program tape preparation and assembly services are available, a very high percentage of programming time is consumed in program assembling and editing. But no one is providing a text editing program that is both convenient to use and powerful enough for the experienced user.

Text editors are based upon the simple principle of reading a chunk of text into computer memory, modifying it through keyboard commands, and then outputting a corrected file. Most editors force the user to modify text at the line level — if a line of text has a single character error in it, the user must type the entire line over again. In addition, the actual addressing or locating of the errant text is a difficult process with the text editors available today. To overcome these problems, the NOVA text editor is organized around both line and character operations. Single characters, character strings, whole lines and multiple lines may be inserted, deleted or replaced with single keyboard commands. Text is readily located by means of string searches.

One of the programs that has been most neglected by the manufacturers of small computers is the debugging package. The existing packages are very limited in their permissible use of breakpoints. The user is constrained to use a single breakpoint, if any, and severe restrictions are placed upon the use of the breakpoint — it cannot be used with the machine's program interrupt hardware. The NOVA debugging packages allow the simultaneous operation of four breakpoints with no restrictions upon their placement or usage. The debugger also offers the traditional operations of memory examination and modification, binary punch-out, memory searches and dumps.

The physical construction of the NOVA opens several possibilities for the use of the debugger that have not been available before. Since the NOVA's 5¼" high enclosure can accommodate up to 16K words of core memory in addition to I/O interfaces, the OEM user has the ability to simply plug-in an additional memory module in which the debugging package may reside while checking out his program. The memory module can then be removed before shipment of the machine and applications program. If program bugs are uncovered during field usage of the system, a memory module with the debugger in it can be installed, the source of the bug identified and corrected, and the module removed.



Herbert J. Richman, Vice President, Data General Corporation

Configuring your system

Small computers should come in any size you wish. The NOVA does.

A general purpose NOVA configuration has a central processor, console, power supply, 4096 sixteen-bit words of core memory and an interface for Teletype. But you can configure your system smaller than this. You can have as little as 1024 words of memory. You can make this memory read-only and remove the display console and have the least expensive computer controller you can buy.

Or you can gracefully expand the basic NOVA. You see, a big part of the total price of a small computer system is in the cost of packaging the system. The NOVA has as much room for expansion in the basic configuration as most customers will need.

The NOVA rack mount version takes up only 5¼ inches of a standard 19 inch rack. The desk top version is slightly larger and handsomer. Both can contain the same amount of hardware.

The NOVA, rack or desk model, contains space for seven printed circuit subassemblies. Each of these subassemblies are 15″ x 15″ with a 200 pin connector on one end and handles for insertion and removal on the other end.

The boards which are inserted in these slots may be any one of several system components. Two slots are used for the central processor. One slot is used for each memory module (4K, 2K, or 1K) added. One slot can contain an I/O option card which contains the control logic for several standard peripheral devices.

Thus, for a 4K system with Teletype, four slots are used and three are available for additional options, memory, and customer designed and built logic. Both the memory bus and the I/O bus are available at these slots so options or memory may be added by simply plugging in the appropriate sub-assembly. No extra wiring is required.

A 5¼″ tall NOVA expansion cabinet can be added to the basic NOVA. It also has the memory bus and I/O bus pre-wired to the slots using printed circuit wiring.

Service, pricing, and delivery

NOVA is backed by a generous guarantee and trained Data General service personnel. These regionally-based service personnel can set up just the kind of service arrangement you need.

They can recommend a back-up of NOVA spare parts and sub-assemblies so that you may never have to call him. You will be able to have your NOVA repaired through the mails at Data General's factory and not lose a minute of computer time.

When you get your NOVA we will teach you how to use it. Comprehensive training classes are available for NOVA programming and maintenance. You will receive complete documentation: User's Handbook, Interface and Installation, Software and Maintenance. You will also receive a documentation up-dating service including an expanding NOVA software library.

We were after a better performing computer and NOVA's performance exceeds that of any machine in its price class in every benchmark we have run. And we deliberately chose benchmarks that competitive manufacturers had been using to demonstrate the superiority of their particular machines.

We were after a lower priced computer and the price for the NOVA is very much lower for comparative configurations of each and every competitive machine — from the most stripped-down controller to general purpose systems with mass storage.

We know that over half the small computers purchased are purchased in quantity by the same customer. And that it costs us less to sell and service one hundred computers to one customer than to one hundred individual customers.

We also know that the way to lower the price of computers is to manufacture in volume.

So we are offering by far the best quantity and OEM discounts ever offered for small computers.

We believe the only way to go in this small computer business is big. So we're starting out to manufacture hundreds of NOVA's our first year. Our rapidly increasing rate of production will equal or exceed the fastest delivery rate of any small computer in the industry.

Soon we will be able to deliver NOVAs as fast as they are ordered. Until then, it's first order, first machine.

NOVA specifications and instructions

SPECIFICATIONS

NOVA is a 16-bit word general purpose computer. It has four accumulators, two of which may be used as index registers. It offers a choice of core or read-only memory of 1K, 2K, 4K, 8K, and up to 32K 16-bit words (or twice that many 8-bit bytes). NOVA comes in desk top console or a 5¼" tall standard rack mount package. Both the desk and rack versions can hold up to 20K 16-bit words of memory or interfaces for a large number of peripheral devices. NOVA has the most flexible I/O facility ever built into a machine of its class. It includes a high-speed Data Channel and automatic interrupt source identification as standard equipment.

Electrical specifications

Power Requirements
90 to 250 volts, 40 to 440 Hz single phase power capable of supplying approximately 5 amperes.

Receptacle required to receive standard three wire plug.

Power Dissipation
400 watts.

I/O Bus Levels
Ground and +5 volts (standard TTL integrated circuit logic levels).

Environmental specifications

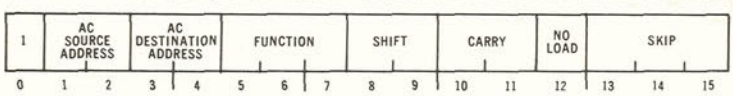
Operating Temperature
0°C to 50°C.

Relative Humidity
To 90%.

INSTRUCTIONS

ARITHMETIC AND LOGICAL INSTRUCTIONS

An instruction that has a 1 in bit 0 performs one of eight arithmetic and logical functions as specified by bits 5–7 of the instruction word. The function, which may be anything from a simple move to a subtraction, always uses the contents of the accumulator specified by bits 1 and 2; and if a second operand is required, it comes from the accumulator addressed by bits 3 and 4.



The instruction also supplies a carry bit to the shifter with the result. Bits 10 and 11 specify a base value to be used in determining the carry bit. The instruction supplies either this value or its complement depending upon both the function being performed and the result it generates. The mnemonics and bit configurations and the base values they select are as follows.

Mnemonic	Bits 10–11	Base valve for carry bit
	00	Current state of carry
Z	01	Zero
C	10	Complement of current state of carry
O	11	One

The three logical functions simply supply the listed value as the carry bit to the shifter. The five arithmetic functions supply the complement of the base value if the operation produces a carry out of bit 0; otherwise they supply the value given. The carry bit 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.

The 17-bit word consisting of the carry bit and the 16-bit result is operated on by the shifter as specified by bits 8 and 9.

Mnemonic	Bits 8–9	Shift operation
	00	None
L	01	Left rotate one place. Bit 0 is rotated into the carry position, the carry bit into bit 15
R	10	Right rotate one place. Bit 15 is rotated into the carry position, the carry bit into bit 0
S	11	Swap the halves of the 16-bit result. The carry bit is not affected

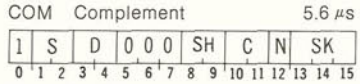
The shifter output is also tested for a skip according to the condition specified by bits 13–15. The processor skips the next instruction if the specified condition is satisfied.

Mnemonic	Bits 13–15	Skip function
	0	Never Skip
SKP	1	Always Skip
SZC	2	Skip on Zero Carry
SNC	3	Skip on Nonzero Carry
SZR	4	Skip on Zero Result
SNR	5	Skip on Nonzero Result
SEZ	6	Skip if Either Carry or Result is Zero
SBN	7	Skip if Both Carry and Result are Nonzero

Arithmetic and Logical Functions

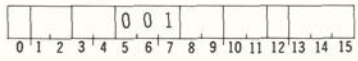
The eight functions are selected by bits 5–7 of the instruction word. For convenience the accumulators addressed by the S and D parts of the instruction are referred to as ACS and ACD.

COM Complement 5.6 μs



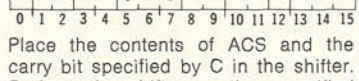
Place the (logical) complement of the word from ACS and the carry bit specified by C in the shifter. Perform the shift operation specified by SH. Load the shifter output in carry and ACD unless N is 1. Skip the next instruction if the shifter output satisfies the condition specified by SK.

NEG Negate 5.6 μs



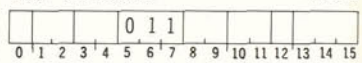
Place the twos complement of the number from ACS into the shifter. Perform the shift operation specified by SH. Load the shifter output in carry and ACD unless N is 1. Skip the next instruction if the shifter output satisfies the condition specified by SK.

MOV Move 5.6 μs



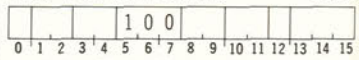
Place the contents of ACS and the carry bit specified by C in the shifter. Perform the shift operation specified by SH. Load the shifter output in carry and ACD unless N is 1. Skip the next instruction if the shifter output satisfies the condition specified by SK.

INC Increment 5.6 μs



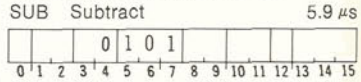
Add 1 to the number from ACS and place the result in the shifter. Perform the shift operation specified by SH. Load the shifter output in carry and ACD unless N is 1. Skip the next instruction if the shifter output satisfies the condition specified by SK.

ADC Add Complement 5.9 μs



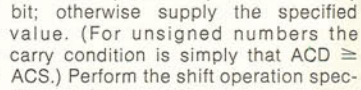
Add the (logical) complement of the number from ACS to the number from ACD, and place the result in the shifter. Perform the shift operation specified by SH. Load the shifter output in carry and ACD unless N is 1. Skip the next instruction if the shifter output satisfies the condition specified by SK.

SUB Subtract 5.9 μs



Subtract by adding the twos complement of the number from ACS to the number from ACD, and place the result in the shifter. If the signs of the operands are the same and ACD ≥ ACS, or the signs differ and ACD is negative, supply the complement of the value specified by C as the carry bit; otherwise supply the specified value. (For unsigned numbers the carry condition is simply that ACD ≥ ACS.) Perform the shift operation specified by SH. Load the shifter output in carry and ACD unless N is 1. Skip the next instruction if the shifter output satisfies the condition specified by SK.

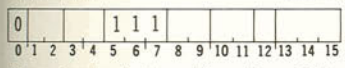
ADD Add 5.9 μs



Add the number from ACS to the number from ACD, and place the result in the shifter. If both summands are negative, or their signs differ and their magnitudes are equal or the positive one is the greater in magnitude, supply the complement of the value specified by C as the carry bit; otherwise

supply the specified value. (For unsigned numbers the carry condition is simply that the sum is $\geq 2^{16}$.) Perform the shift operation specified by SH. Load the shifter output in carry and ACD unless N is 1. Skip the next instruction if the shifter output satisfies the condition specified by SK.

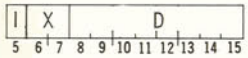
AND And 5.9 μs



Place the logical and function of the word from ACS and the word from ACD in the shifter. Supply the value specified by C as the carry bit. Perform the shift operation specified by SH. Load the shifter output in carry and ACD unless N is 1. Skip the next instruction if the shifter output satisfies the condition specified by SK.

MEMORY REFERENCE INSTRUCTIONS

Bits 5–15 have the same format in every memory reference instruction whether the effective address is used for storage or retrieval of an operand or to alter program flow. Bit 5 is the indirect bit, bits 6 and 7 are the index

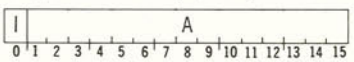


bits, and bits 8–15 are the displacement. The effective address E of the instruction depends on the values of I, X, and D. If X is 00, D addresses one of the first 256 memory locations, i.e., D is a memory address in the range 00000–00377. This group of locations is referred to as page zero.

If X is nonzero, D is a displacement that is used to produce a memory address by adding it to the contents of the register specified by X. The displacement is a signed binary integer in twos complement notation. Bit 8 is the sign (0 positive, 1 negative), and the integer is in the octal range —200 to +177 (decimal —128 to +127). If X is 01, the instruction addresses a location relative to its own position, i.e., D is added to the address in PC, which is the address of the instruction being executed. This is referred to as relative addressing. If X is 10 or 11 respectively, it selects AC2 or AC3 as a base register to which D is added.

X	Derivation of address
00	Page zero addressing. D is an address in the range 00000–00377.
01	Relative addressing. D is a signed displacement (—200 to +177) that is added to the address in PC.
10	Base register addressing. D is a signed displacement (—200 to +177) that is added to the address in AC2.
11	Base register addressing. D is a signed displacement (—200 to +177) that is added to the address in AC3.

If I is 0, addressing is direct, and the address already determined from X and D is the effective address used in the execution of the instruction. Thus a memory reference instruction can directly address 1024 locations: 256 in page zero, and three sets of 256 in the octal range 200 less than to 177 greater than the addresses in PC, AC2 and AC3. If I is 1, addressing is indirect, and the processor retrieves another address from the location specified by the address already determined. In this new word bit 0 is the indirect bit: bits 1–15 are the effective address if bit 0 is 0; otherwise they specify a location



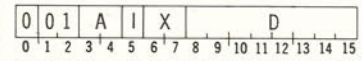
for yet another level of address retrieval. This process continues until some referenced location is found with a 0 in bit 0; bits 1–15 of this location are the effective address E.

If at any level in the effective address calculation an address word is fetched from locations 00020–00037, it is automatically incremented or decremented by one, and the new value is both written back in memory and used either as the effective address or for the next step in the calculation depending on whether bit 0 is 0 or 1. Addresses taken from locations 00020–00027 are incremented, those from locations 00030–00037 are decremented.

Move Data Instructions

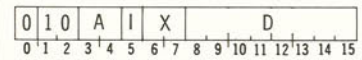
These two instructions move data between memory and the accumulators. In the descriptions of all memory reference instructions, *E* represents the effective address. The time given in the top line is for direct addressing, in page zero or relative to PC. Base register addressing requires an additional .3 μs; indirect addressing requires one extra memory cycle time per level; autoincrementing and autodecrementing require no extra time.

LDA Load Accumulator 5.2 μs



Load the contents of location E into accumulator A. The contents of E are unaffected, the original contents of A are lost.

STA Store Accumulator 5.5 μs

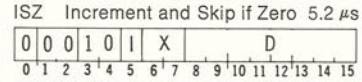


Store the contents of accumulator A in location E. The contents of A are unaffected, the original contents of E are lost.

Modify Memory Instructions

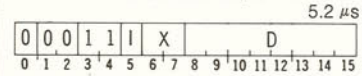
These two instructions alter a memory location and test the result for a skip. They are used to count loop iterations or successively modify a word for a series of operations.

ISZ Increment and Skip if Zero 5.2 μs



Add 1 to the contents of location E and place the result back in E. Skip the next instruction in sequence if the result is zero.

DSZ Decrement and Skip if Zero 5.2 μs

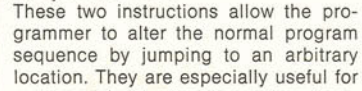


Subtract 1 from the contents of location E and place the result back in E. Skip the next instruction in sequence if the result is zero.

Jump Instructions

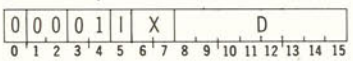
These two instructions allow the programmer to alter the normal program sequence by jumping to an arbitrary location. They are especially useful for calling and returning from subroutines.

JMP Jump 2.6 μs



Load E into PC. Take the next instruction from location E and continue sequential operation from there.

JSR Jump to Subroutine 3.5 μs



Load an address one greater than that in PC into AC3 (hence AC3 receives the address of the location following the JSR instruction). Load E into PC. Take the instruction from location E and continue sequential operation from there. The original contents of AC3 are lost.

INPUT-OUTPUT INSTRUCTIONS

Instructions in the in-out class govern all transfers of data to and from the peripheral equipment, and also perform various operations within the processor. An instruction in this class is designated by 011 in bits 0–2. Bits 10–15 select the device that is to respond to the instruction. The format thus allows for 64 codes of which 62 can be used to address devices (octal 01–76). The code 00 is not used, and 77 is used for a number of special functions including reading the console data switches and controlling the program interrupt.

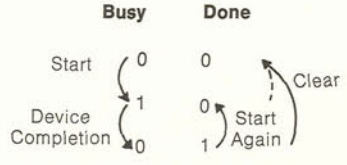
Every device has a 6-bit device selection network, an Interrupt Disable flag, and Busy and Done flags. The selection network decodes bits 10–15 of the instruction so that only the addressed device responds to signals sent by the processor over the in-out bus. The Busy and Done flags together denote the basic state of the device. When both are clear the device is idle. To place the device in operation, the program sets Busy. If the device will be used for output, the program must give a data-out instruction that sends the first unit of data — a word or character depending on how the device handles information. (The word "output" used without qualification always refers to the transfer of data from the processor to the peripheral equipment; "input" refers to the transfer in the opposite direction.) When the device has processed a unit of data, it clears Busy and sets Done to indicate that it is ready to receive new data for output, or that it has data ready for input. In the former case the program would

respond with a data-out instruction to send more data; in the latter with a data-in instruction to bring in the data that is ready. If the Interrupt Disable flag is clear, the setting of Done signals the program by requesting an interrupt; if the program has set Interrupt Disable, then it must keep testing Done or Busy to determine when the device is ready.

In all in-out instructions bits 8 and 9 either control or sense Busy and Done. In those instructions in which bits 8 and 9 specify a control function, the mnemonics and bit configurations and the functions they select are as follows.

Mnemonic	Bits 8–9	Control function
	00	None
S	01	Start the device by clearing Done and setting Busy.
C	10	Clear both Busy and Done, idling the device.
P	11	Pulse the special in-out bus control line — the effect, if any, depends on the device.

The overall sequence of Busy and Done states is determined by both the program and the internal operation of the device.



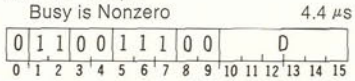
The data-in or data-out instruction that the program gives in response to the setting of Done can also restart the device. When all the data has been transferred the program generally clears Done so the device neither requests further interrupts nor appears to be in use, but this is not necessary. Busy and Done both set is a meaningless situation.

Bits 5–9 specify the complete function to be performed. If there is no transfer (bits 5–7 all alike), bits 3 and 4 are ignored and bits 8 and 9 may specify a control function or a skip condition.

NIO No I/O Transfer 4.4 μs

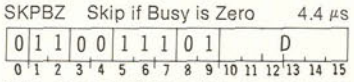
Perform the control function specified by *F* in device *D*.

SKPBN Skip if Busy is Nonzero 4.4 μs



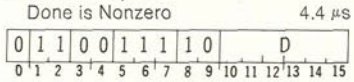
Skip the next instruction in sequence if the Busy flag in device *D* is 1.

SKPBZ Skip if Busy is Zero 4.4 μs



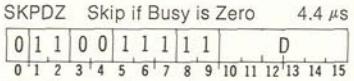
Skip the next instruction in sequence if the Busy flag in device *D* is 0.

SKPDN Skip if Done is Nonzero 4.4 μs



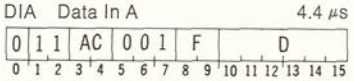
Skip the next instruction in sequence if the Done flag in device *D* is 1.

SKPDZ Skip if Busy is Zero 4.4 μs



Skip the next instruction in sequence if the Done flag in device *D* is 0.

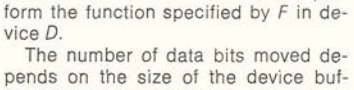
DIA Data In A 4.4 μs



Move the contents of the A buffer in device *D* to accumulator *AC*, and perform the function specified by *F* in device *D*.

The number of data bits moved depends on the size of the device buffer, its mode of operation, etc. Bits in *AC* that do not receive data are cleared.

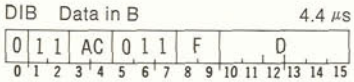
DOA Data Out A 4.7 μs



Send the contents of accumulator *AC* to the A buffer in device *D*, and perform the function specified by *F* in device *D*.

The amount of data actually accepted by the device depends on the size of its buffer, its mode of operation, etc. The original contents of *AC* are unaffected.

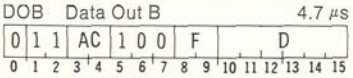
DIB Data in B 4.4 μs



Move the contents of the B buffer in device *D* to accumulator *AC*, and perform the function specified by *F* in device *D*.

The number of data bits moved depends on the size of the device buffer, its mode of operation, etc. Bits in *AC* that do not receive data are cleared.

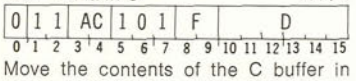
DOB Data Out B 4.7 μs



Send the contents of accumulator *AC* to the B buffer in device *D*, and perform the function specified by *F* in device *D*.

The amount of data actually accepted by the device depends on the size of its buffer, its mode of operation, etc. The original contents of *AC* are unaffected.

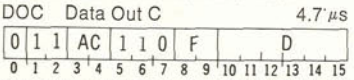
DIC Data in C 4.4 μs



Move the contents of the C buffer in device *D* to accumulator *AC*, and perform the function specified by *F* in device *D*.

The number of data bits moved depends on the size of the device buffer, its mode of operation, etc. Bits in *AC* that do not receive data are cleared.

DOC Data Out C 4.7 μs



Send the contents of accumulator *AC* to the C buffer in device *D*, and perform the function specified by *F* in device *D*.

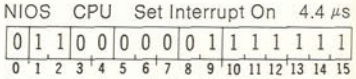
The amount of data actually accepted by the device depends on the size of its buffer, its mode of operation, etc. The original contents of *AC* are unaffected.

Special Code-77 Functions

In-out instructions with the code 77 in bits 10–15 perform a number of special functions rather than controlling a specific device. In all but the skip instructions bits 8 and 9 are used to turn the interrupt on and off. The mnemonics are the same as those for controlling Busy and Done in I/O devices, but with code 77 they select the following special functions.

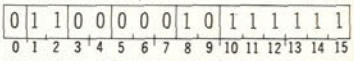
Mnemonic	Function
S	Set the Interrupt On flag to enable the processor to respond to interrupt requests.
C	Clear the Interrupt On flag to prevent the processor from responding to interrupt requests.
P	None.

NIOS CPU Set Interrupt On 4.4 μs



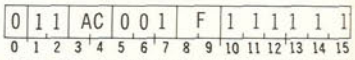
Set the Interrupt On flag to allow the processor to respond to interrupt requests.

NIOC CPU Clear Interrupt On 4.4 μs



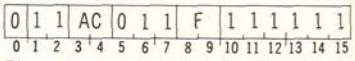
Clear the Interrupt On flag to prevent the processor from responding to interrupt requests.

DIA *AC*,CPU Data In A, Processor 4.4 μs



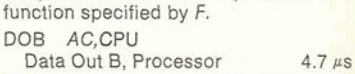
Read the contents of the console data switches into accumulator *AC*, and perform the function specified by *F*.

DIB *AC*,CPU Data In B, Processor 4.4 μs



Place in accumulator *AC* the device code of the first device on the bus that is requesting an interrupt ("first" means the one that is physically closest to the processor on the bus.) Perform the function specified by *F*.

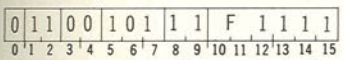
DOB *AC*,CPU Data Out B, Processor 4.7 μs



Set up the Interrupt Disable flags in the devices according to the mask in accumulator *AC*. For this purpose each device is connected to a given data

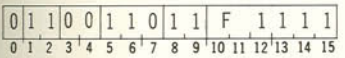
line, and its flag is set or cleared as the corresponding bit in the mask is 1 or 0. Perform the function specified by *F*.

DIC 0,CPU
Data In C, Processor 4.4 μs



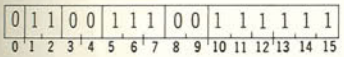
Clear the control flipflops, including Busy, Done and Interrupt Disable, in all devices connected to the bus. Perform the function specified by *F*.

DOC 0,CPU
Data Out C, Processor 4.7 μs



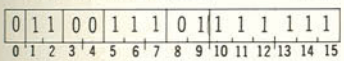
Perform the function specified by *F*. and then halt the processor.

SKPBN CPU Skip if
Busy is Nonzero, Processor 4.4 μs



Skip the next instruction in sequence if the Interrupt On flag is 1.

SKPBZ CPU Skip if
Busy is Zero, Processor 4.4 μs



Skip the next instruction in sequence if Interrupt On is 0.

The assembler recognizes a number of convenient mnemonics for instructions with device code 77.

Mnemonic	Meaning	Mnemonic Equivalent		Octal Equivalent
READS	Read Switches	DIA	—,CPU	060477
IORST	IO Reset	DICC	0,CPU	062677
HALT	Halt	DOC	0,CPU	063077
INTEN	Interrupt Enable	NIOS	CPU	060177
INTDS	Interrupt Disable	NIOC	CPU	060277
INTA	Interrupt Acknowledge	DIB	—,CPU	061477
MSKO	Mask Out	DOB	—,CPU	062077

NOVA Options

1K Read-Only Memory Module (1024 words of 16 bits each)
Includes 1K words (2K bytes) of Read-Only Memory which is interchangeable with the alterable core memory with no wiring modifications required. The contents of this module can be either standard programs or special customer specified programs.

NOVA Central Processor — Rack Mountable
Includes central processor, console (with lock), high-speed data channel, power supply, five subassembly slots available in basic frame. This basic frame is rack mountable in a standard 19″ rack with slides.

NOVA Central Processor — Table Top Model
Includes central processor, console (with lock), high-speed data channel power supply, five subassembly slots available in basic frame. This basic frame is enclosed in a cabinet designed for table top use.

4K Core Memory (4096 words of 16-bits each)
Includes 4K words (8K bytes) of memory with all necessary electronics mounted on a single subassembly (15″x15″ printed circuit card) which can be plugged directly into one of the slots in the basic frame with no wiring modifications required.

2K Core Memory (2049 words of 16 bits each)
Includes 2K words (4K bytes) of memory mounted on a single subassembly (15″x15″ printed circuit card) which can be plugged directly into one of the slots in the basic frame with no wiring modifications required.

1K Core Memory (1024 words of 16 bits each)
Includes 1K words (2K bytes) of memory mounted on a single subassembly (15″x15″ printed circuit card) which can be plugged directly into one of the slots in the basic frame with no wiring modifications required.

Expansion Enclosure
This item is a basic frame with a power supply and slots to mount seven (7) subassemblies. This unit is typically mounted directly above the central processor frame and is used to mount additional memory (alterable core or Read Only), additional I/O controllers, or special customer designed hardware.

Power Monitor and Auto Restart
Provides power level detection and a flag which is attached to the Program Interrupt and can be sensed by the program. Its function is to allow the program to become aware of an imminent power failure so it can provide for an orderly shut down. The program automatically restarts at location 0.

Real-Time Clock
This option provides a flag which can be enabled by the program to provide a program interrupt at a fixed frequency. Either the AC line or a crystal clock may be specified as the time source.

Teletype Input/Output Interface
This option provides an interface to any one of the Teletype models listed below:

Teletype ASR33
Keyboard/printer, 8 channel reader/punch, 10 char./sec.

Teletype KSR33
Keyboard/printer, 10 char./sec.

Teletype KSR35
Keyboard/printer, 10 char./sec.

Teletype KSR37
Keyboard/printer (upper/lower case), 15 char./sec.

High-Speed Perforated Tape Reader and Control
Paper tape reader and control which senses eight channel, fan fold perforated Mylar or paper tape photoelectrically at 150 or 300 characters per second.

High-Speed Perforated Tape Punch and Control
BRPE11 Punch and Control which punches eight channel fan fold paper tape at 63.3 characters per second.

Cathode Ray Tube Display and Control
This equipment includes a Tektronix Type 602 5″ Rectangular Display and a control interface operating in a point plotting mode (256 points x 256 points) using the Data Channel for display refresh.

Medium-Speed Line Printer and Control
This option includes a Potter Medium-Speed Line Printer Type HSP3502 and control. This is a chain printer with a full ASCII interface including paper advance (LF, FF) characters.

Card Reader and Control
Soroban SCCR card reader and control which operates at 225, 400 or 600 cards per minute.

Incremental Plotter
Plotters and control interfaces to units made by two different manufacturers. 1) The California Computer Products 500 Series units (Drum or Flat Bed Style). 2) The Houston Instruments Complot DP-1 Digital Plotter which uses "Z fold" style paper.

In addition, the following devices will be available: magnetic disk, incremental tape, IBM compatible magnetic tape, and a complete line of A/D and D/A equipment.

Printed U.S.A. © Nov.1968 PL1-20M

DATA GENERAL CORPORATION

Southboro, Massachusetts 01772

Thank you for your interest in NOVA.

NOVA is a 16-bit word general purpose computer. It has four accumulators, two of which may be used as index registers. It offers a choice of core or read-only memory of 1K, 2K, 4K, 8K, and up to 32K 16-bit words (or twice that many 8-bit bytes). NOVA comes in a desk top console or a 5¼" tall standard rack mount package. Both the desk and rack versions hold up to 20K 16-bit words of memory or interfaces for a large number of peripheral devices. NOVA has the most flexible I/O facility ever built into a machine of its class. It includes a high-speed Data Channel and automatic interrupt source identification as standard equipment.

A 4096 sixteen-bit word configuration with Teletype interface costs $7950. But the NOVA can be configured much smaller or bigger than this. And there are quite liberal quantity discounts — up to 40%.

If you wish additional information and a chance to study "How to use the NOVA," a rather complete reference manual, just drop the attached post card in the mail.

Southboro, Massachusetts 01772

I want more information about the NOVA.

Name _____ Title: _____

Company/Organization _____

Address _____

Phone _____

My application interest is _____

FIRST CLASS PERMIT
No. 26
Southboro
Mass. 01772

BUSINESS REPLY MAIL

No Postage Necessary If Mailed In The United States

Postage will be paid by:

Data General Corporation
Southboro, Massachusetts 01772