Nanodata Model QM-1 Central Processing Unit

The QM-1 is a high-speed general purpose digital computer that operates under two levels of microprogram control. The unique design of the QM-1 supports a system of software-created user levels, whereby users at different levels approach architecture, machine language, and programming in ways most suited to their own specific requirements of the hardware.

THE QM-1 CONTROL HIERARCHY

In the QM-1, a two-level design smooths the machine definition process over two stages, achieving the advantages of both Horizontal and Vertical control:

Machine instructions in Main Store are executed by (and defined by) microprograms in Control Store, under Vertical Control.

Microinstructions in Control Store are in turn executed by (and defined by) Nanoprograms in Nanostore, under Horizontal Control.

An illustration of this concept is shown in Figure 1.

In particular, Control Store is a fully general-purpose Read/Write store, hence it is feasible, for some applications, to approach QM-1 Control Store as the primary program store of the machine, executing programs which can regard the passive Main Store as a secondary storage unit.

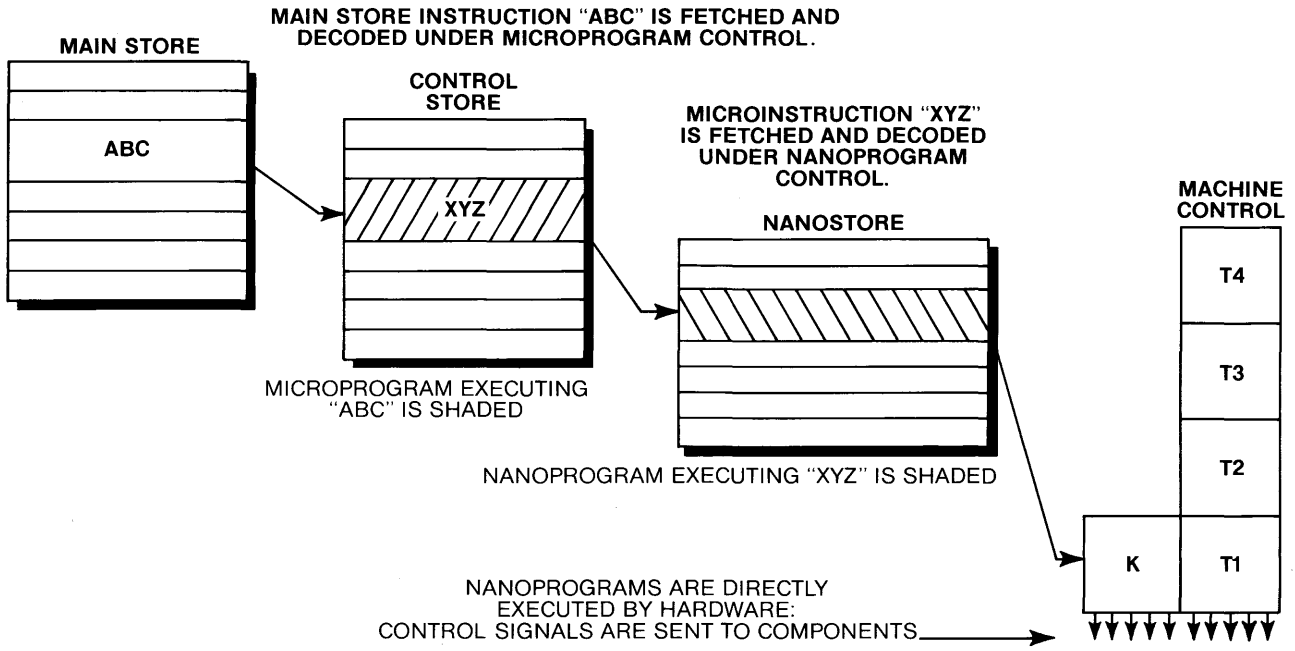


Figure 1. QM-1 Control Hierarchy — Example of Two Level Emulation

USER AND MACHINE HIERARCHIES

The design of the QM-1 lends itself to a system of "virtual machines" arranged in a hierarchy of levels. Each level is supported by the machine below, and in turn supports the machine above. Once a given machine is defined by suitable software (or "firmware"), its implementation — i.e., the nature of that software structure — is transparent to the user of that machine. For example, after suitable nanoprogramming is done to define a "mico-machine," the very existence of Nanostore is irrelevant to the micro-machine user.

HARDWARE LEVEL

The basic hardware components of the QM-1 include several banks of registers; a system of three stores; arithmetic, boolean, and shift components; and twelve independent buses. Bus connections between the components are programmable and may be changed as often as required to best fit the current task. All these units may be exercised independently, allowing a high degree of parallelism.

Complete control over the hardware is provided by a 360-bit word read from the dynamically writable Nanostore; the active Nanoword provides a sequence of four Machine State Vectors, each of which drives the individual machine components and their interconnections during a machine clock period.

NANO-MACHINE LEVEL

Nanoprogramming is the process of defining a set of such control sequences to implement Microinstructions executed at the next level. The opcode of a vertically formatted Microinstruction, read from Control Store, is used to select the entry point in Nanostore at which to begin executing the defined Nanoprogram. The Microinstruction Set used may be either that defined by NANODATA (with possible user modifications/extensions for the current task) or that defined by the user; the NANODATA supplied microlanguage is accompanied by systems software to support I/O and process management.

MICRO-MACHINE LEVEL

Since microinstructions reside in the fully Readable/Writable Control Store, microprogramming can be used to define the application directly. Due to the flexibility provided at the Nano level, a variety of micro-machines may be defined to efficiently match varied applications. The micro-machine can then be viewed as a conventional machine with a customized instruction set and a 150 Nanosecond memory.

MAIN-STORE-MACHINE LEVEL

For many applications, the above number of levels will be sufficient; applications software may be written in the defined microlanguage, executing out of Control Store at very high speeds. For those applications in which another level of flexibility is desired, however, microprogramming in Control Store may be used to define the architecture and instruction set for software in Main Store. At the micro-level, Main Store is viewed simply as a passive general-purpose data store; the process is one of classical emulation.

The Hardware Level Users Manual provides complete functional specifications of the QM-1, and thus defines the "nano-machine" available to the hardware-level user. Many users will be concerned with the machine at this most fundamental level. The NANODATA Systems Software staff, for example, approaches the machine at this level. Already developed nanocode for support of a general purpose micro-instruction set is available, however, and when such appropriate software, including both systems support functions and any one of several microlanguage definitions, is included in the QM-1, the micro-level user can program the machine without being concerned with the structure beneath.

Specifications

TECHNOLOGY
- T^2L — MSI
- Basic cycle time 80ns

PROCESSOR
- Instruction Word Length
  - Nanostore 360 Bits
  - Control Store Variable
  - Main Store Variable
- Data Word Length
  - Control Store Groups of 18 Bits
  - Main Store Groups of 18 Bits
- General Registers
  - 32 18 Bit Hardware
  - 12 6 Bit Hardware
- Indexing Registers
  - 32 18 Bit Hardware
  - 32 6 Bit Hardware
- Special Purpose and I/O Registers
  - 32 18 Bit Hardware
  - 28 6 Bit Hardware
- Direct Addressing
  - 1024K 18 Bit Words of Main Store
  - 40K 18 Bit Words of Control Store
  - 1K 360 Bit Words of Nanostore
- Arithmetic
  - 2 Fully parallel, 18 bit wide, arithmetic/logic units providing 16 boolean and 32 arithmetic transformations.
  - 1 Six bit wide arithmetic/logic unit providing 16 boolean and arithmetic transformations. (Optional)
  - A Matrix shift unit, 36 bits wide, capable of performing circular, logical, or sign extending shifts or any value in either right or left direction. An optional 32 bit wide matrix is available to facilitate 8/16/32 bit circular shifts.
- Bussing
  - 14 Major Independent 18 Bit Data Paths
  - 7 Major Independent 6 Bit Data Paths

Specifications (Continued)

I/O MODES
- Programmed Transfer — Eight 18 Bit MUX Channels
  - Block transfer up to 1960K Bytes/sec.
- Direct memory access — Five 18 Bit MUX Channels
  - 1 million 18 Bit Words/sec or 2 million 9 Bit Words/sec.
- Priority Interrupts — Identification of up to 512 hardware levels with automatic device identification and vectoring.
  - Interrupt latency time 480ns.

MEMORIES

Main Store	18 Bits + 2 parity bits Cycle Time 750ns
Control Store	18 Bits Access Time 75ns
Nanostore	360 Bits Access Time 75ns

INPUT POWER

208 Volts, 3 phase	60HZ @ 7KVA
Optionally	50HZ

ENVIRONMENT

Operating Temperature	60°F to 80°F (15°C to 26°C)
Operating Humidity	40% to 60% Non-condensing
Heat Dissipation	25,000 BTU/Hr.

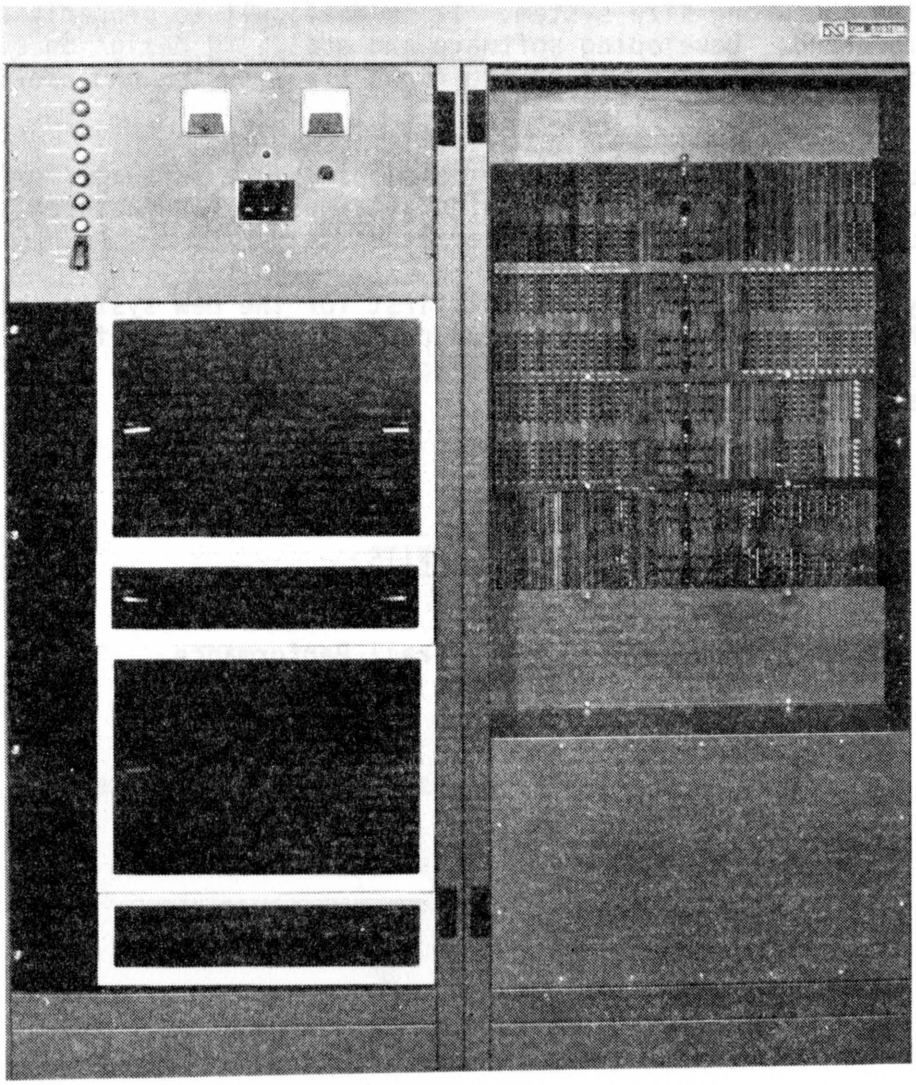
DIMENSIONS

Height	61.75" (156.8 cm)
Width	55.5" (141.0 cm)
Depth	27" (68.6 cm)
Weight	1500 lbs. (680.4 kg)

NANODATA CORPORATION
2457 Wehrle Drive • Williamsville, N.Y. 14221 • (716) 631-5880

QM-1

APPLICATION/BENEFITS AVAILABLE SUPPORT CONFIGURATIONS



NANODATA CORPORATION
2457 Wehrle Drive Williamsville, New York 14221 (716) 631-5880

DEVELOP PROGRAMS ON OUR SYSTEM FOR THE EXPENSIVE SYSTEM YOU PLAN TO BUY

Here's why • • •

- Program development takes time. Use our system to write and debug programs, in order to defer major system expenditures until they are warranted. You'll save dollars in the long run on rentals, depreciation, and cost of money.
- The passage of time increases competitive pricing options open to you and reduces your vulnerability to destandardization. Our system allows you to initiate a major development effort without incurring substantial costs or risks.
- Avoid buying the wrong size system. It is difficult to properly size a system beforehand. Developing software and analyzing design on our system will aid immeasurably in ordering the right sized system and provide you true cost data before the fact.
- Insure you order the right system by developing some software for two or three manufacturers on our system rather than using manufacturer's benchmarks results as sole selection criteria.
- Use our system to develop software at first for the new system and later to replace the old system. Use it to run programs that are difficult to convert. Chances are the maintenance dollars saved are enough to offset the entire cost of our system.

USER BENEFITS

- Increase Productivity and Overall Performance
- Stretch an Education or R & D Budget
- On Our System Develop Programs for Expensive Systems

INCREASE PRODUCTIVITY AND OVERALL PERFORMANCE WITH OUR SYSTEM

Here's how • • •

Our system, because of its unique features, allows users to develop software for other systems more easily and determine bottlenecks faster. It is invaluable in developing software for minis, and will allow the user to determine in advance the upper limits to which a mini can be expanded or pushed.

On-line software development on some computers is costly in terms of hardware or reduced system performance. Our system can be used for that software development.

Our hardware can be used to support many types of systems, from an IBM 370 to a one of a kind airborne computer. In fact, our system has already emulated a military computer a year before the real computer was built.

Use our system to tie a network together, as a node in a distributed processing environment emulating the central site CPU. Or use it as a translation processor allowing different types of computers to communicate effectively with each other.

STRETCH AN EDUCATION OR R & D BUDGET

Here's how • • •

- Education tool for basic or advanced study in programming, design, and architecture of any system or many systems.
- Develop a special purpose system with unique instruction repertoire -- APL processor, DATA BASE Management Processor, Formula machines for math, physics and other sciences.
- Test and validate new computer designs for any system from microprocessors to multi-processor fourth generation systems.
- Save the expense of ruggedized equipment in a benign environment.
- Use our system over and over, for entirely different purposes, without ever requiring hardware modifications. Your R & D efforts can be kept entirely in-house and secret.
- Our system is hundreds of times more cost effective than simulation techniques on other systems.

HERE'S HOW ONE NANODATA CUSTOMER IS USING THEIR SYSTEM

APPLICATIONS

- ARCHITECTURE DESIGN FOR REAL TIME AND NEAR REAL TIME COMMUNICATION, COMMAND AND CONTROL SYSTEMS
- DEVELOP AND VALIDATE MICROPROCESSOR FIRMWARE AND SOFTWARE
- DEVELOP SOFTWARE FOR OPERATIONAL COMPUTERS WHICH ARE INACCESSIBLE BECAUSE THEY LACK I/O OR ARE ONE OF A KIND
- DEVELOP EMULATORS FOR OBSOLETE, HARD TO MAINTAIN COMPUTERS
- SPECIFY COMPUTER ARCHITECTURES BASED ON SOFTWARE DESIGN
- SOFTWARE AND FIRMWARE VERIFICATION AND VALIDATION

RESEARCH

- COMPUTER ARCHITECTURE DEFINITION EMULATION COMPILER
- DISTRIBUTED DATA BASE SIMULATION FACILITY
- NETWORK DESIGN ANALYSIS
- SATELLITE GRAPHICS DESIGN
- COMPUTER ARCHITECTURE STUDIES FOR ARTIFICIAL INTELLIGENCE AND FAULT-TOLERANT COMPUTING
- HIGH ORDER LANGUAGE PROCESSORS AND SPECIAL PURPOSE PROCESSORS
- MICROPROCESSOR DESIGN

POPULAR SYSTEM EMULATORS AVAILABLE FROM NANODATA THROUGH A PROGRAM LICENSE AGREEMENT

IBM 360 (DOS and OS)

DG NOVA Series

DEC PDP 11/10

EMULATORS DEVELOPED BY NANODATA FOR CUSTOMERS

IBM 7094 with Modified I/O

Trident Missile Fire Control Computer System

CDC 160A

CDC 200UT

EMULATORS UNDER CONSIDERATION

DEC PDP 11/70

DG Eclipse

EMULATORS COMPLETED OR BEING DEVELOPED BY CUSTOMERS

Systems

AN/UYK 20

INTEL 8080

UNIVAC 1106

RCA SCP-234 (MARC 1)

DELCO Magic 352

MIX

DEC PDP 11/40

High Level Language Machines

SIMPL - Q

BLAISE (Extended PASCAL)

APL

Concurrent PASCAL

SYSTEM SOFTWARE

NANODATA CONTROL SYSTEM (NCS)

NCS is a basic operating system supporting data and program files in a disk storage environment, with absolute minimum memory overhead.

NCS is an overlay-oriented system. Each system overlay performs a specific task for the console operator, a user program, or for other system subroutines.

Those wishing to modify NCS can do so with the assembly capability provided by the QM microassembler.

MULTI MICROMACHINE (MULTI)

The MULTI Micromachine is a multi-purpose, microprogramming architecture. It was designed to be the base instruction set, whereby with the appropriate specialized extensions for any target architecture, the task of microprogramming an emulator can be easily and quickly accomplished. The architecture has 18/36-bit or 16/32-bit (Single/Double) data width, and 18 or 36-bit instruction width. The 80 instructions encompass the categories of control store operations, arithmetic operations, shifting operations, main store operations, branching and testing operations, special control and data operations, and system support operations. All of the basic operations in an emulation environment can be programmed with this instruction set. Extensions to this set are common in existing emulators but only necessary for speed and brevity at the micro level.

PROGRAMMABLE RUN-TIME OPERATOR (PROD)

The Programmable Run-Time Operator display is designed to provide the emulator designer with a flexible yet uniform control base on which to build his emulator. Its primary purpose is to allow the operator to control the emulated CPU. PROD also provides trace/debug capabilities for the emulated hardware system and display/debug capabilities for the microprogrammer.

PROD consists of five logical sections. A task manager who oversees the execution of the emulator, the debug/display package, the console emulation routines (which provide the operator a method of controlling the emulated CPU's console), special support interfaces, and instrumentation supports. Each section is easily modified to suit the particular target while the nucleus is universal and provides a working base on which a wide range of facilities may be added.

SYSTEM SOFTWARE

QM MICROASSEMBLER (MICRO)

The QM Microassembler is a main store program which provides for assembly of QM/system control store microprograms.

MICRO is extendable and can be restructured, dynamically, to conform to any of the microprogramming architectures defined for execution on the QM-1.

MICRO is a conventional two-pass assembler with special provision for the definition of microinstruction operation codes, formats and constants when used for the assembly of microprograms.

TASK CONTROL PROGRAM (TCP)

TCP supports multiple co-resident control store tasks. Tasks are controlled in a hierarchical manner. Task 0 is the supervisory control program. Contained within the supervisor task are all input/output control functions, as well as task scheduling routines.

Task 1 is activated immediately after system initialization and becomes the primary system task. Task 1 maintains communications with the operator console, while also controlling the execution of the object program, or emulator, at task level 2.

QM NANOASSEMBLER (NA)

The QM Nanoassembler is a main store program for translating source programs into modules for loading QM-1 nanostore.

NA frees the QM-1 user from explicitely specifying each of the many bits in each nanoword. It allows, instead, general symbols and transfers to be specified by the user and then expands these into the necessary bit configurations.

NA can generate a full binary output file for use in loading nanostore or it can generate a definition file to be used as input to the QM microassembler.

Special Hardware Devices Available from Nanodata

IBM 360 Compatible Channel Controller
UNIVAC 1100 Peripheral Interface Channel

Education Courses Offered

One-Day Microprogramming and QM-1 System Overview
Two-Week QM-1 Programming Class
Two-Week System Engineering Class
One-Week Advanced Systems and Methods

User Group Membership (meets annually)

President: Professor Walter A. Burkhard
University of California, S.D./
Computer Science Division/Applied Physics Dept.

Participating Members:
University of Alberta/Dept. of Computer Sciences/
Edmonton
U.S.A.F./Griffiss AFB/Rome Air Development Center
U.S. Army/Harry Diamond Labs/Adelphi
U.S. Navy/Naval Surface Weapons Center/Dahlgren
TRW/Systems Group/Redondo Beach
Martin Marietta/Aerospace Div./Denver
McDonnell Douglas/McAuto Div./Astronautics Co./
Huntington Beach

Maintenance

Contract maintenance is available on a yearly renewable basis. Principal period of maintenance consists of eight contiguous hours (9 a.m. - 5 p.m.) daily, Monday-Friday. Normal response is within 24 hours.

Contract Programming

Prices based on level of effort and potential resale value.

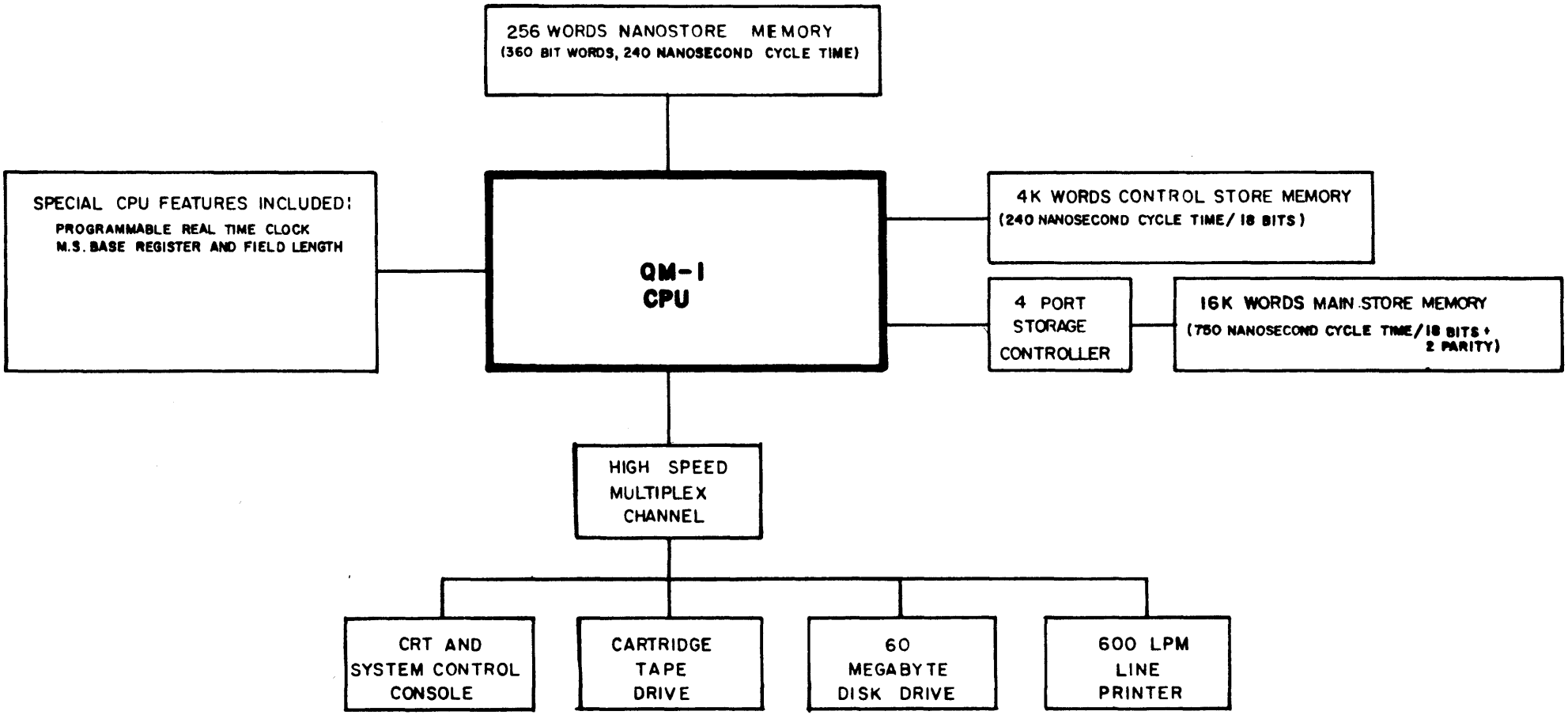
NANODATA MANUALS

No.	Description
M1	QM-1 Hardware Level User's Manual
M2	QM Micro - QM Microassembler Reference Manual
M3	MULTI Micromachine Description
M4	PROD User's Guide - Programmable Run-Time Operator Display
M6	Microprogramming and the QM System
M7	TASK - Task Control Program Overview
M13	Environment and Space Requirements for QM System
M14	QM Interfacing Manual
M15	Loader Services Program (LSP)
M8	QM-NCS - Systems Operation Guide (NOVA, U1100, etc.)
M12	Operating the 360 Emulator
M19	PDP 11/10 Emulator Control Program Summary
M17	QM-1 200 UT User's Guide
M9	BLAISE on the QM-1
M10	Line Printer System
M11	Data Communication Controller
M16	Precision Real Time Clock
M18	Magnetic Tape System

NANODATA PRODUCT BULLETINS

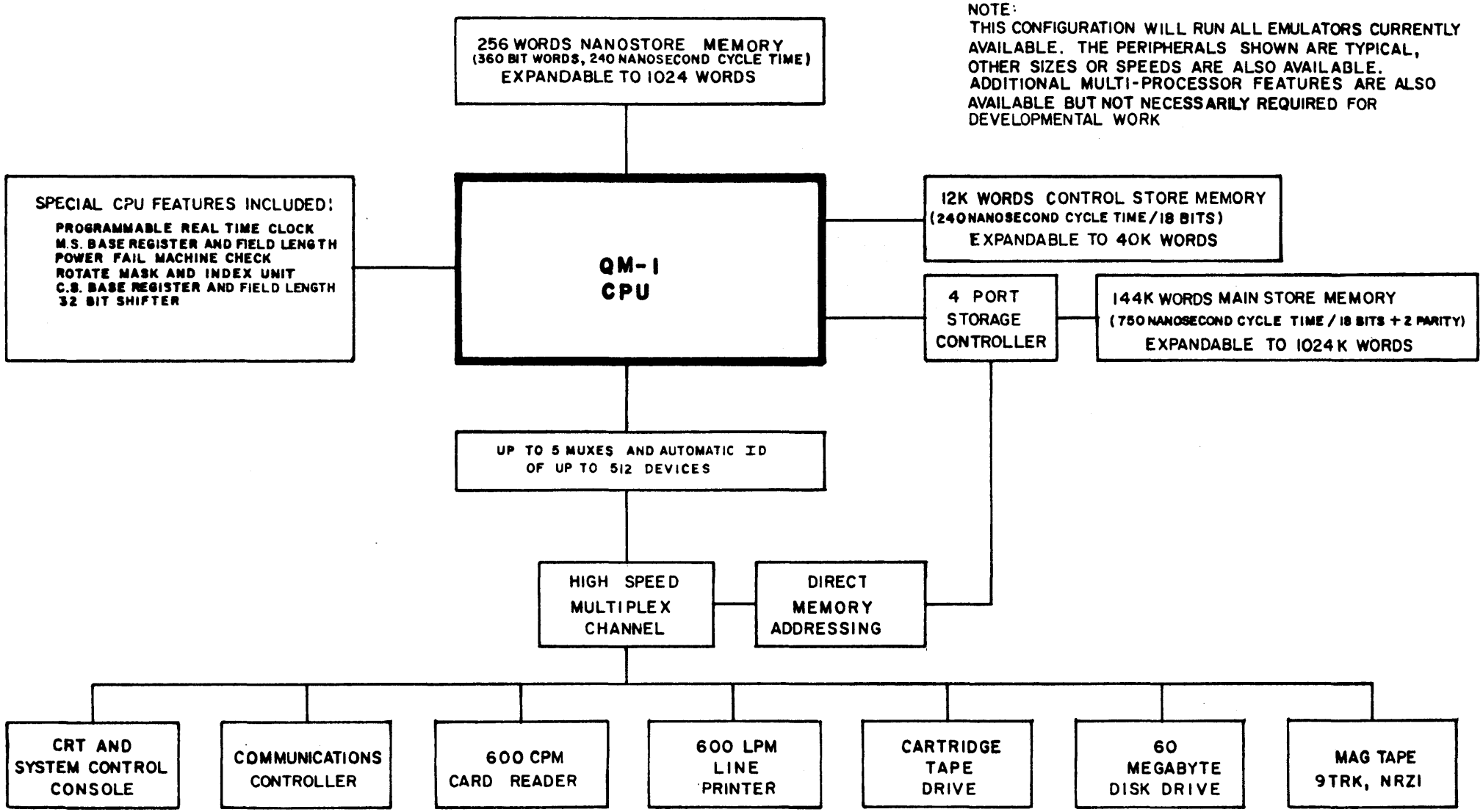
Description
Model QM-1 Central Processing Unit
Rotate, Mask & Index Unit (RMI)
360 Channel Interface
1100 Channel Interface
CRT Terminal
Cartridge Tape System
Disk Systems (12 or 60 meg byte drives)
Chain Printers (300, 400, 600 and 800 LPM)
Drum Printer
Card Reader/Punch
Card Reader (200, 600 and 1,000 CPM)
Magnetic Tape Systems (45-125 IPS, NRZI and PE, 7 and 9 track)
Paper Tape Punch

MINIMUM SYSTEM CONFIGURATION



SELLING PRICE $190,000
MONTHLY MAINTENANCE $1,150
INSTALLATION CHARGE $1,200
SHIPPING CHARGES F O B NANODATA

TYPICAL CUSTOMER CONFIGURATION



NOTE:
THIS CONFIGURATION WILL RUN ALL EMULATORS CURRENTLY AVAILABLE. THE PERIPHERALS SHOWN ARE TYPICAL, OTHER SIZES OR SPEEDS ARE ALSO AVAILABLE. ADDITIONAL MULTI-PROCESSOR FEATURES ARE ALSO AVAILABLE BUT NOT NECESSARILY REQUIRED FOR DEVELOPMENTAL WORK

SELLING PRICE $287,000
MONTHLY MAINTENANCE $1,800
INSTALLATION CHARGE $1,200
SHIPPING CHARGES F O B NANODATA

"Imagine a...programmers machine...designed to provide all the tools for debugging....logical breakpoints, displays, etc..... that the programmer...always wished he had...but that previously required rewiring the underlying hardware.....By means of a microprogrammed emulation....such an 'extended' machine architecture is a workable, practicable, efficient alternative to blinking-lights, toggle-switched, and other paraphenalia of the nardware engineer....."

C.W. Flink, U.S. Navy, MICRO-10 PROCEEDINGS, Oct. 5, 1977.

For more information on the NANODATA Corp. QM/1 please contact:

San Diego, CA

Robert C. Boe
714 226 8502

Williamsville, NY

Michael C. Senft
716 631 5880

MICROPROGRAMMING

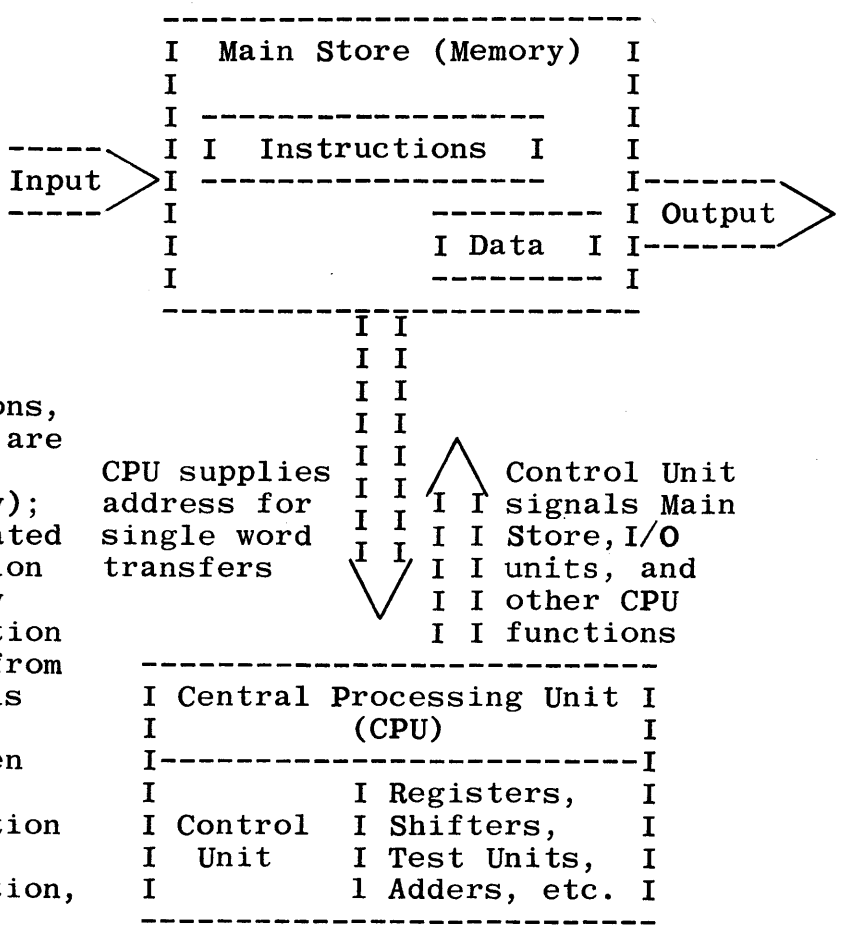
AND THE

QM-1

Introduction To Microprogramming And the QM-1

Every programmable device, or "machine", possesses an architecture and an instruction set. The architecture is its system of components and their interconnection; in the case of a computer, architectures are described in terms of stores, registers, arithmetic-logic units, data paths, etc. A machine instruction is a command which causes elements of the architecture to operate in some predetermined manner; the instruction set of a machine is simply a list of all instructions which the machine recognizes.

Using these broad definitions and the simplified model of a computer shown in Figure A, a discussion of three phases of the "instruction sequence" provides a basic explanation of computer operation.



Block Diagram of a Computer
Figure A

Instruction Fetch

Sequences of machine instructions, in the form of binary numbers, are typically stored in contiguous locations in Main Store (memory); instruction execution is initiated by fetching a machine instruction from a given location in memory and placing it into an Instruction Register. The memory address from which to fetch an instruction is contained in an Instruction Location Counter Register, often called a Program Counter; part of the effect of every instruction is to update this register to point to the successor instruction, and then to begin the memory fetch for the next sequential instruction.

Instruction Decode

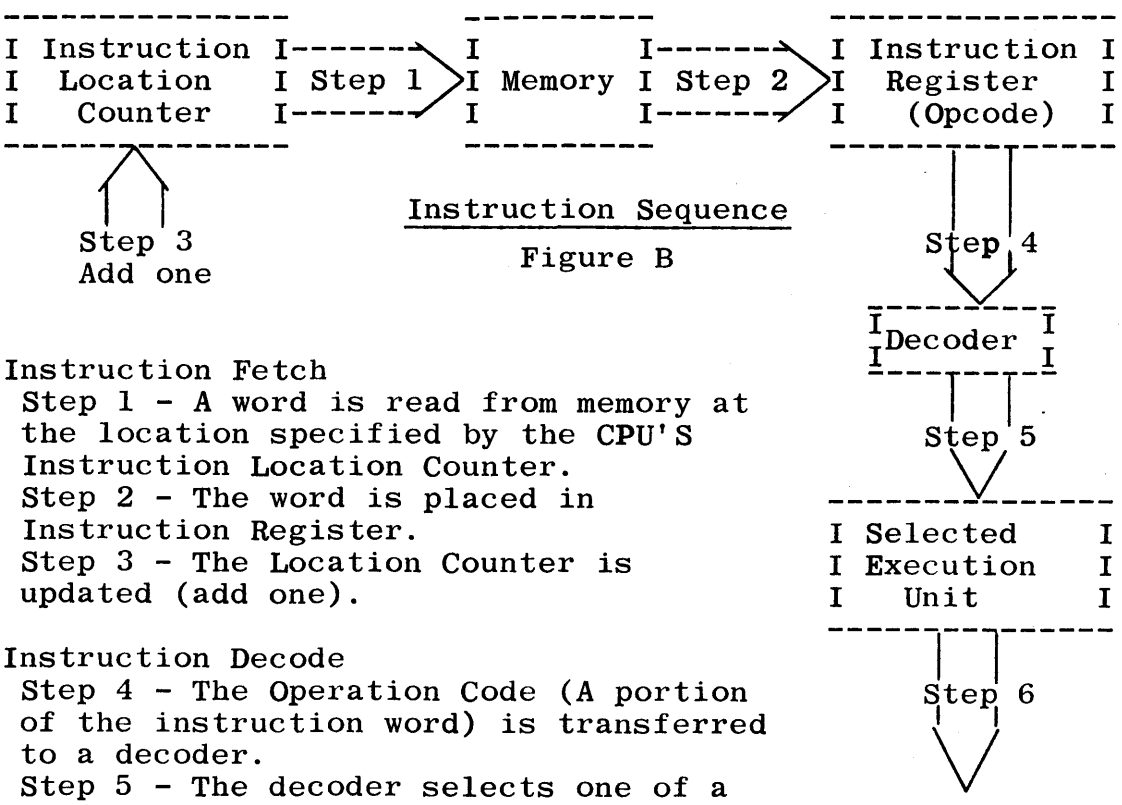
A portion of the contents of the Instruction Register is designated as the Operation Code. This binary number is decoded by the Control Unit to select among a number of modules, each of which is responsible for accomplishing the effect of one of the instructions in the computer's instruction set. As will be shown later, the method of decoding and the nature of these modules is critical to the definition of Microprogramming.

Instruction Execution

The ultimate effect of any instruction-execution module is the generation of electrical signals to the various computer components.

Basic Instruction Sequence

These three phases of Instruction Fetch, Decode, and Execute, form the basic Instruction Sequence (or "instruction cycle"). After initial start-up, all computers follow an instruction sequence similar to that illustrated in Figure B.



Instruction Sequence

Figure B

Instruction Fetch
Step 1 - A word is read from memory at the location specified by the CPU'S Instruction Location Counter.
Step 2 - The word is placed in Instruction Register.
Step 3 - The Location Counter is updated (add one).

Instruction Decode
Step 4 - The Operation Code (A portion of the instruction word) is transferred to a decoder.
Step 5 - The decoder selects one of a number of execution plans.

Instruction Execute
Step 6 - Carry out execution plan which may include data fetch, data manipulation, data store, repeatedly.
End of Sequence - Do next instruction fetch (Step 1.).

Microprogrammed Control

The final phase is of particular interest here. The electrical signals which the control unit sends to the architectural components are the most basic, or "primitive", commands in the computer; these signals have effects such as opening and closing gates (for example, to transfer register contents), initiating memory cycles, and setting individual bits. In fact, the Instruction Sequence itself is under the control of such primitive operations; an implicit effect of every machine instruction is the execution of the next Instruction Sequence.

Only rarely do machine instructions correspond to a single architectural primitive; most machine instructions result in the generation of a number of primitives, frequently arranged in a time sequence. For some instructions, the arrangement of primitives can be fairly complex. An example is a Multiply instruction on a machine which has only an adding component, the adder must be used iteratively, and the internal plan of the instruction resembles a computer program.

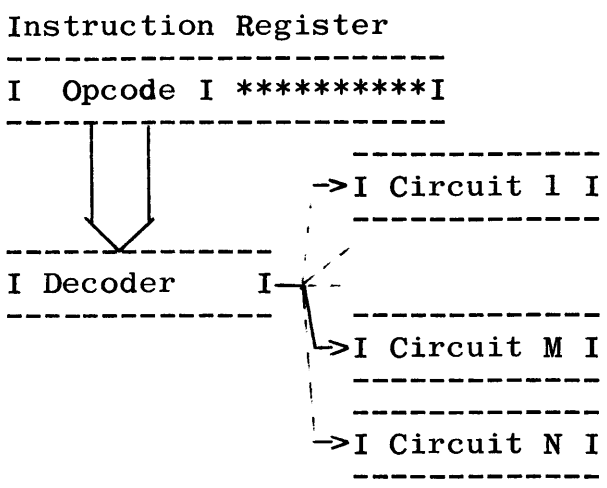
The later observation suggests an implementation of the primitive signal control function. In the conventional, or "hard-wired" computer, a hardware decoding of the relevant portion of the instruction word selects one of several logic circuits, each of which is responsible for generating and sequencing the primitive signals of a given machine instruction. If, however, the primitive control functions are regarded as "microoperations", then a "microprogram" can be written to plan the flow of an instruction. The steps of this microprogram can then be implemented as primitive commands executing out of a fast-access store, such as semiconductor memory. (Execution of such commands is simple to accomplish, since the microoperations correspond directly to architectural functions.)

Using a microprogrammed approach to machine instruction implementation, the instruction-decode step of machine operation changes: rather than decoding the operation-code portion of the instruction to select one of several hardware modules, this binary number is used directly as an address, or pointer, into the microprogram store ("control store"); the location so defined is programmed as the entry point of the microprogram which implements the original machine instruction. This process is illustrated in Figure C.

The Operation Code of a Machine Instruction determines the arrangement and timing of the signals which control movement of data between Memory, CPU Registers, Arithmetic-Logic Units and other hardware facilities.

Hard-Wired Computer

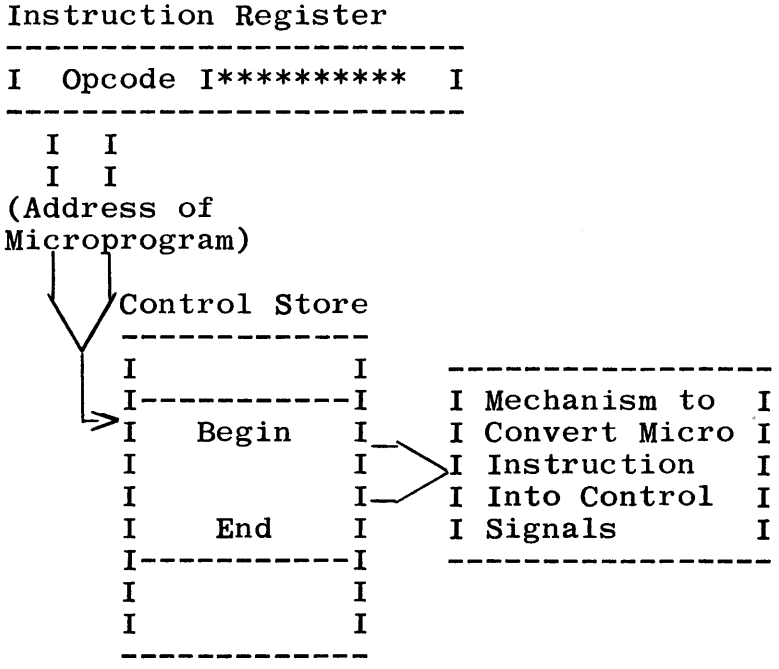
In a conventional (hard-wired) computer, the opcode is decoded and used to select among logic circuits which provide the control signals within the computer.



Machine Instructions--- and hence the functional nature of the computer as seen by the programmer - are determined by the hardware machine designer.

Microprogrammed Computer

In a microprogrammed computer, the opcode is used as an address (pointer) into a fast "control store." The microprogram starting at that address has been written to provide the control signals.



Machine Instructions -- and hence the functional nature of the computer as seen by the programmer -- are determined by the microprogrammer and may be redefined as readily as the control store may be reprogrammed. If control store is writable, the user can microprogram at his convenience, modifying his machine at computer speeds instead of "soldering iron" speeds.

Figure C - Instruction Decode Schemes

Uses of Microprogramming

With the previously defined model of microprogram machine control, we can now examine the uses and advantages of microprogramming. The strongest single justification for microprogramming lies in the current disparity between the speed of main store (core memory) and the speed of currently available logical components. For example, more than 10 sets of primitive functions may be executed in the time taken to read one word from core memory. Thus time exists for more than 10 control store steps to implement a main store instruction. This large ratio makes possible a significant increase in the power of the instructions defined at the higher level over those required in the underlying hardware. For this reason, microprogramming is now common in many computers.

Microprogramming provides other advantages as well. Since microprogramming in control store serves to define the computer as seen at the conventional level, the flexibility of microprogramming may be used to vary the machine defined. Many of the advantages that result are tabulated in Figure D, and with a writable control store any number of the uses may be achieved in the same machine hardware by dynamically altering the contents of control store to match the needs.

From a fully flexible microprogrammed design arises the full gammet of uses, from the historical one of defining a common instruction set to range across a variety of architectures, to the common use of emulating another computer, to the ultimate use of control store being utilized as the user level, with main store acting as some type of file, paging, or higher-level language system.

Microprogramming May Be Used To	Advantages
1. Define a computers instruction set independent of the basic hardware development. This was the most common early use.	A) Separates the instruction definition from the hardware specification. B) Permits matching memory speed to logic speeds when a large difference exists.
2. Cause the hardware to function as another (pre-existing) computer. This is the common definition of emulation.	A) Emulated machines software may be used without modification thus preserving possibly large software investments. B) Host computer system may be faster or less expensive than original machine. C) Several machines may be emulated at different times, on same hardware host.
3. Emulate another computer, but with extra instructions and/or special features.	Increased efficiency: functions requiring complex and time consuming software may be performed directly on the machine, as a single (special) instruction. Examples: * floating point arithmetic * operating system functions * any programmed procedure commonly used in a given application.
4. Create a special-purpose computer to meet the needs of a particular environment.	A) Microprogram development is easier, faster, and less expensive than hardware development, and is performed by personnel typically closer to end needs than hardware personnel. B) Result can be modified easily when necessary, as needs change. C) When application is phased out, host hardware remains usable.
5. Write user programs in control store, with main store used as a fast message buffer, page backup, file storage, etc.	Very fast processing times are possible for suitable applications. Less hardware may be necessary to do the job since the hardware is used directly.

Figure D

Horizontal and Vertical Control

The designer of a machine with microprogrammed control faces an immediate decision as to the format of microinstructions to be used in the machine. He may choose to use a wide, unstructured microword, usually called a horizontal microinstruction:

```
-------------------------------------------  Each bit is
.....Horizontal Microinstrution.....  independent
-------------------------------------------  of other bits.
```

When executed, each bit in a horizontal microinstruction results in a control signal to a hardware component. This is generally found in more powerful machines. The microinstruction may run to 100 or more bits (the IBM 360/50 uses a microinstruction 90 bits wide).

Or the designer may choose a highly encoded microinstruction packed into a much smaller word. The word contains a micro-opcode and several other encoded fields. For this reason, it is often referred to as a vertical microinstruction:

```
                          ------------------------  Several bits form
Vertical Microinstruction  Micro-Opcode / XXX XXXX  an encoded field.
                          ------------------------
```

When executed, the micro-opcode of a vertical microinstruction selects a sequence of control signals, similar to the operation of a machine instruction opcode but at a lower level (simpler sequences are invoked). Vertical microinstructions are much shorter (the IBM 360/25 has a 16 bit microinstruction).

Each scheme for microprogrammed control offer certain advantages. A choice involves evaluation of many trade-offs. Some of the factors are tabulated in Figure E.

Conclusions:

> Horizontal microinstructions are preferable to vertical microinstructions for flexibility and parallelism, but they are more difficult to program, require larger amounts of expensive storage and are limited in what time sequences may be programmed.

The QM-1 has been designed to make available the advantages of each scheme of microprogrammed control and to avoid the disadvantages inherent in each. The unique features of the QM-1 that make this possible will be examined next.

Trade-Offs Between Horizontal And Vertical Control — Figure E

Horizontal Microinstructions...	Vertical Microinstructions...
Allow ultimate flexibility in control, since each signal (bit) may be individually selected by the microprogrammer.	Provide a limited selection of control patterns; the number of possibilities depends upon the width of the Micro-Opcode.
May be executed simply by gating them to a register, to which signal lines are attached directly.	Require execution machinery similar to (but simpler than) that required to execute machine instructions.
Allow parallel operation of hardware components.	Typically specify "single-thread" operations.
Are relatively difficult to program.	Are relatively simple to program.
Must be executed frequently, since they exercise each hardware component at most once.	May specify a time-sequence of control signals, so they may be executed less frequently.
Are wide, typically on the order of 100 bits.	Require only enough bits to contain the Micro-Opcode and perhaps some parameters -- typically 8 to 18 bits.
The last two items imply that storage of enough horizontal microinstructions to run a reasonably powerful emulation may be expensive in number of bits.	The last two items imply that storage of enough vertical microinstructions to run a reasonably powerful emulation may be inexpensive in number of bits.

The QM-1 Control Hierarchy

In the QM-1, a two-level design smooths the machine definition process over two stages, achieving the advantages of both Horizontal and Vertical control:

Machine instructions in Main Store are executed by (and defined by) microprograms in Control Store, under Vertical control.

Microinstructions in Control Store are in turn executed by (and defined by) Nanoprograms in Nanostore, under Horizontal control.

An illustration of this concept is shown in Figure F

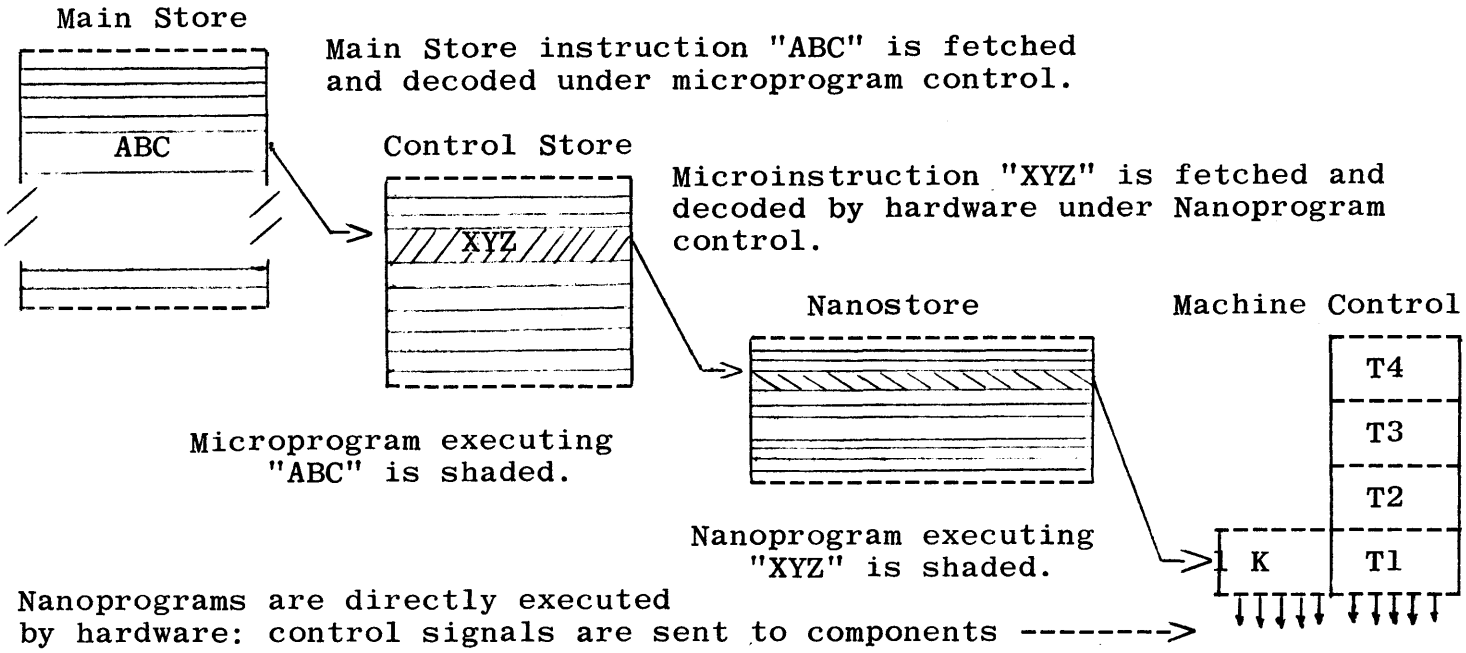


Figure F QM-1 Control Hierarchy Example of Two Level Emulation

This unique control hierarchy takes advantage of the best features of both horizontal and vertical control as summarized in Figure G. In addition, flexible time sequencing is possible at both levels. And most important, both control store and nanostore are fully writable semiconductor memories, so that the QM-1 user can take advantage of all possible flexibility in the system by dynamic reprogramming.

In particular, control store is a fully general-purpose Read/Write store, hence it is feasible, for some applications, to approach QM-1 Control Store as the primary program store of the machine, executing programs which can regard the passive Main Store as a secondary storage unit.

Control Hierarchy Dimensional Trade-Offs — Figure G

At Highest Level		At Lowest Level
End user has system simple to program.	———	Hardware designer has system directly implementable.
Generalized indirect control. Powerful (high level) instructions.	———	Absolute direct control. Primitive (low level) functions.
Meaning of main store contents fully redefinable.	———	Meaning of control signals fixed in hardware.
Large memory available.	———	Small store required.
Low cost/bit.	———	Fast operation.

User and Machine Hierarchies

The design of the QM-1 lends itself to a system of "virtual machines" arranged in a hierarchy of levels. Each level is supported by the machine below, and in turn supports the machine above. Once a given machine is defined by suitable software (or "firmware"), its implementation -- I.E., the nature of that software structure -- is transparent to the user of that machine. For example, after suitable nanoprogramming is done to define a "micro-machine", the very existence of Nanostore is irrelevant to the Micro-Machine user.

Hardware Level

The basic hardware components of the QM-1 include several banks of registers; a system of three stores; arithmetic, boolean, and shift components; and twelve independent buses. Bus connections between the components are programmable and may be changed as often as required to best fit the current task. All these units may be exercised independently, allowing a high degree of parallelism.

Complete control over the hardware is provided by a 360-bit word read from the dynamically writable Nanostore; the active Nanoward provides a sequence of four Machine State Vectors, each of which drives the individual machine components and their interconnections during a machine clock period of 75 Nanoseconds.

Nano-Machine Level

Nanoprogramming is the process of defining a set of such control sequences to implement Microinstructions executed at the next level. The opcode of a vertically formatted Microinstruction, read from Control Store, is used to select the entry point in Nanostore at which to begin executing the defining Nanoprogram. The Microinstruction Set used may be either that defined by NANODATA (with possible user modifications/extensions for the current task) or that defined by the user; the NANODATA-supplied micro-language is accompanied by systems software to support I/O and process management.

Micro-Machine Level

Since microinstructions reside in the fully Readable /Writable Control Store, microprogramming can be used to define the application directly. Due to the flexibility provided at the Nano level, a variety of Micro-machines may be defined to efficiently match the application. The Micro-machine can then be viewed as a conventional machine with a customized instruction set and a 150 Nanosecond memory.

Main-Store-Machine Level

For many applications, the above number of levels will be sufficient; applications software may be written in the defined microlanguage, executing out of Control Store at very high speeds. For those applications in which another level of flexibility is desired, however, microprogramming in Control Store may be used to define the architecture and instruction set for software in Main Store. At the micro level, Main Store is viewed simply as a passive general-purpose data store; the process is one of classical emulation.

As indicated previously, the Hardware Level Users Manual provides complete functional specifications of the QM-1, and thus defines the "nano-machine" available to the hardware-level user. Many users will be concerned with the machine at this most fundamental level. The NANODATA Systems Software staff, for example, approaches the machine at this level. Already developed nanocode for support of a general purpose micro-instruction set is available, however and when such appropriate software, including both systems support functions and any one of several micro language definitions, is included in the QM-1, the micro-level user can program the machine without being concerned with the structure beneath.

The QM-1 Architecture

The architecture of the QM-1 is specifically designed to span the levels of programming with the flexibility needed at the lowest level, and the power necessary for the highest. The hardware drawing on the next page gives an idea of the extent to which the architecture is flexible. It shows the data paths of the QM-1 computer, and that group of components, which make up the controls and functional units as viewed at the lowest level by the nanoprogrammer. This drawing can be an insight into what is available at this level, however, without having to program at this level. The parallelism which this level is capable of providing for the micro level and the main store level, through the implementation of specialized support for a specific application, greatly extends the power of each level. The micro architecture will, in general, be a major extension of the QM-1 architecture, and the Main Store architecture can be likewise; e.g., a virtual machine at either level can be designed with a large number of general purpose registers (probably residing in control store). The range of feasible virtual machine definitions is limited only by the ingenuity of the designer, and the efficiency considerations of the emulation process; stack machine architectures, sophisticated arithmetic processors, and "wideword" or "bit addressable" machines are only a few examples of the kinds of possibilities which are all well within range.