QM-1
HARDWARE LEVEL
USER'S MANUAL

NANODATA CORPORATION
2457 Wehrle Drive Williamsville, New York 14221

```
  QQQ     M     M                1
 Q   Q    MM   MM               11
Q     Q   M M M M              1 1
Q     Q   M  M  M                1
Q     Q   M     M   -------      1
Q     Q   M     M                1
Q   Q Q   M     M                1
 Q   Q    M     M                1
  QQQ Q   M     M              11111
```

H A R D W A R E   L E V E L
U S E R'S   M A N U A L

Second Edition

Revision 2

August 31, 1974

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

Copyright c 1972 NANODATA CORPORATION

TABLE OF CONTENTS

1. SCOPE AND PURPOSE

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 present document explains these concepts and defines the QM-1 as it appears to the "hardware-level" user.

The "hardware-level" user approaches a programming interface whose functional parts correspond to the facilities provided by the physical QM-1 computer itself, without any restrictions to the full generality of the hardware imposed by pre-definition of the contents of any of the machines control memories. Even the contents of the Read-Only Memories, included for machine bootstrap and diagnostic purposes, may be programmed by this user. The Hardware-Level User's Manual, while not an engineering or maintenance document, is thus oriented toward the QM-1 user whose purpose is to define his own computer starting at the lowest possible level.

2. 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 interconnections; 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 2a, a discussion of three phases of the "instruction sequence" provides a basic explanation of computer operation.

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.

```
                --------------------------
                I   MAIN STORE (MEMORY)   I
                I                         I
                I ----------------        I
       -----\   I I INSTRUCTIONS I        I ------\
       INPUT >  I ----------------        I OUTPUT >
       -----/   I                -------- I ------/
                I                I DATA I I
                I                -------- I
                I                         I
                --------------------------
                         I  I   /\
           CPU supplies  I  I  /  \ Control unit
           address for   I  I  I  I signals main
           single word   I  I  I  I store, IO units,
           transfers     I  I  I  I and other CPU
                         \  /  I  I functions
                          \/   I  I
                --------------------------
                I CENTRAL PROCESSING UNIT I
                I          (CPU)          I
                I-------------------------
                I           I Registers,  I
                I  CONTROL  I Shifters,   I
                I   UNIT    I Test Units, I
                I           I Adders, etc.I
                --------------------------
```

BLOCK DIAGRAM OF A COMPUTER
Figure 2A

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.

2.1 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 2B.

```
---------------           ----------          ---------------         -----------
I INSTRUCTION I-------\ I          I-------\ I INSTRUCTION I-------\ I         I
I  LOCATION   I Step 1 >I MEMORY I Step 2 >I   REGISTER   I Step 4 >I DECODER I
I   COUNTER   I-------/ I          I-------/ I   (OPCODE)   I-------/ I         I
---------------           ----------          ---------------         -----------
      /\                                                                  I  I
     /  \     Instruction Fetch                                           I  I
     I  I       Step 1 - A word is read from memory at the location    Step 5
   Step 3       specified by the CPU's Instruction Location Counter.    I  I
   Add One      Step 2 - The word is placed in Instruction Register.    \  /
     I  I       Step 3 - The Location Counter is updated (Add One).      \/
              Instruction Decode                                     ------------
                Step 4 - The Operation Code (a portion of the        I SELECTED  I
INSTRUCTION     instruction word) is transferred to a decoder.       I EXECUTION I
 SEQUENCE       Step 5 - The Decoder selects one of a number of      I   UNIT    I
                execution plans.                                     ------------
 Figure 2B    Instruction Execute                                        I  I
                Step 6 - Carry out execution plan which may include    Step 6
                data fetch, data manipulation, data store, repeatedly. \  /
              End of Sequence - Do next Instruction Fetch (Step 1).     \/
```

2.2 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 "micro-operations", 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 micro-operations 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 2c.

```
------------------------------------------------------------------------------
I THE OPERATION CODE OF A MACHINE INSTRUCTION DETERMINES THE ARRANGEMENT AND  I
I TIMING OF THE SIGNALS WHICH CONTROL MOVEMENT OF DATA BETWEEN MEMORY, CPU    I
I REGISTERS, ARITHMETIC-LOGIC UNITS AND OTHER HARDWARE FACILITIES.            I
------------------------------------------------------------------------------
```

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 computer.

```
INSTRUCTION REGISTER
----------------------
I OPCODE I ********* I
----------------------
   I  I
   I  I              -------------
   \  /          -->I CIRCUIT 1 I
    \/               -------------
-----------
I DECODER I--
-----------   \      -------------
               \-->I CIRCUIT M I
                     -------------

                     -------------
                 -->I CIRCUIT N I
                     -------------
```

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

```
============================
I  ALTERNATIVE SCHEMES FOR  I
I    INSTRUCTION DECODE     I
I        Figure 2C          I
============================
```

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.

```
 INSTRUCTION REGISTER
 ----------------------
 I OPCODE I ********* I
 ----------------------
  I  I
  I  I CONTROL STORE
  I  I  -----------
  \  /  I/////////I       ----------------
   \/   I---------I       I Mechanism to I
    --->I BEGIN   I--\ I convert micro I
(address I   .     I   >I instruction   I
   of    I   .     I--/ I into control  I
  micro  I  END    I      I signals       I
 program)I---------I      ----------------
         I/////////I
         I/////////I
         -----------
```

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 (rather than "read-only), the user can microprogram at his convenience, modifying his machine at computer speeds instead of "soldering iron" speeds.

2.3 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 2D.

MICROPROGRAMMING MAY BE USED TO	ADVANTAGES Figure 2D
1. DEFINE A COMPUTERS INSTRUCTION SET independent of the basic hardware development. This was the most common early use.	a) Seperates 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 macnines 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.

Since a fully flexible microprogrammed design also performs the instruction fetch and instruction decode under microprogram control, main store becomes merely a storage area which may (among other things) contain instructions of a higher-level machine. Therefore a fifth use of microprogramming is:

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.

2.4 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 MICROINSTRUCTION.............]   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:

```
                                   --------------------------   Together, several
VERTICAL MICROINSTRUCTION         [ Micro-opcode /  xxx xxxx ]  bits form 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 offers certain advantages. A choice involves evaluation of many trade-offs. Some of the factors are tabulated in Figure 2E.

TRADE-OFFS BETWEEN HORIZONTAL AND VERTICAL CONTROL   Figure 2E

HORIZONTAL MICROINSTRUCTIONS...	VERTICAL MICROINSTRUCTIONS...
Allow ultimate flexibility in control, since each signal (bit)	Provide a limited selection of control patterns; the number of

may be individually selected by the microprogrammer.	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 16 bits.
The last two items imply that storage of enough horizontal microinstructions to run a reasonably powerful emulation may be prohibitively expensive in number of bits.	The last two items imply that storage of enough vertical microinstructions to run a reasonably powerful emulation may be acceptably inexpensive in number of bits.

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.

2.5 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 2F - QM-1 Control Hierarchy.

```
 Main Store
------------   Main Store instruction "ABC" is fetched
I----------I   and decoded under microprogram control.
I----------I
I ABC      I\      Control Store
I----------I \     -------------   Microinstruction "XYZ" is fetched and
I----------I  \    I-----------I   decoded by hardware under nanoprogram
/     .    /   \-->I XYZ //////I   control
                   I///////////I\
                   I-----------I \        Nanostore            Machine Control
                   I-----------I  \   -------------------              --------
                   I-----------I   \  I-----------------I              I  T4  I
                   -------------    \->I/////////////////I\             --------
              Microprogram executing  I/////////////////I \            I  T3  I
                 "ABC" is shaded.     I-----------------I  \           --------
                                      -------------------   \          I  T2  I
                                     Nanoprogram executing   \     -------------
                                       "XYZ" is shaded.     -->I K  I  T1  I
Nanoprograms are directly executed by                          -------------
hardware; control signals are sent to components ------------>   I I I I I I
                                                                 V V V V V V
```

Figure 2F QM-1 CONTROL HIERARCHY EXAMPLE OF TWO LEVEL EMULATION

This unique control hierarchy takes advantages of the best features of both horizontal and vertical control as summarized in Figure 2G. 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 ADVANTAGES

Figure 2G

AT HIGHEST LEVEL		AT LOWEST LEVEL
End User has system simple to program.	<--------------------------->	Hardware Designer has system direct to implement.
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.

3. USER AND MACHINE HIERARCHIES

The design of the QM-1 suggests the use of 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.

Such a machine hierarchy is shown in Figure 3A and described in detail below.

```
 (1) HARDWARE LEVEL          Components              THE QM-1 HIERARCHY
                              /   /   \              ------------------
                             /    \    \
     Logic Design          (by NANODATA)
       defines            /       \      \
                        /QM??      \QM-?  \QM-1
 (2) NANO MACHINES  **               **    ****
                                          /  \  \
                                         /    \  \
     Nanoprogramming                    /      \  \
       defines                         /        \  \
                                      /          \  \
 (3) MICRO MACHINES                 **           **  **
                                   /            / \
                                  /            /   \  Or Microprogramming
     Microprogramming            /            /     \  is used to satisfy
       defines                  /            /       \  Application
                               /            /         \   directly.
 (4) MAIN STORE MACHINES    **          ****           **
                                       / \  \          V
                                      /   \  \         V
     Main Store Programming          /     \  \        V
       to satisfy                   /       \-  \      V
                                   /         \   \     V
 (5) APPLICATIONS                **          **   **   **    Figure 3A
```

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 of 80 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 IO 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 160-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 in Section 1, the purpose of this manual is to provide complete functional specifications of the QM-I, and thus to define 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.

When 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.

Thus this manual is dedicated to that new breed - the NANOPROGRAMMER. Other programmers may have interest in the manual in order to understand the hardware that supports the level at which they write programs; the hardware-level user will find the material in the next two sections essential.

4. QM-1 FUNCTIONAL SPECIFICATIONS, PART I

4.1 GENERAL

Sections 4 and 5 of this manual are a complete functional specification of the QM-1 CPU in two parts. Part II (Section 5) is intended to be used as a a programmer's reference guide, and includes control field mappings and encodings as well as detailed functional description. Part I (Section 4) explains QM-1 machine concepts, architecture, and operations, and provides an overview of the QM-1 and an introduction to Part II.

In order to introduce the machine specifications to the first-time reader, Part I becomes progressively more specific as more of the overall QM-1 structure is revealed. Hence the earlier sections of Part I have more detailed explanations in Part II.

The machine described in Sections 4 and 5 (QM-1 FUNCTIONAL SPECIFICATIONS) is the "hardware QM-1". Its architectural features and controls are those available to the lowest level (nano-) programmer. Nanoprogramming may be usefully viewed as the task of implementing a ("virtual") machine definition for use at the next higher (micro) level. The description of the "micro-machine" will not necessarily resemble that of the QM-1, and in fact may be quite different. Any number of the QM-1 resources may be dedicated to the implementation of the micro-machine definition. The nanoprogrammer can assign several of the 32 general-purpose registers (LOCAL STORE) available to him as special-purpose architectural features of the micro-machine, e.g., instruction register and location counter. The micro-machine architecture will in general be an extension (rather than a restriction) of the QM-1 architecture; for example, a micro-machine may be designed with a large number of general-purpose registers (which the nanoprogrammer would probably map into Control Store). The range of feasible micro-machine definitions is limited only by the ingenuity of the nanoprogrammer and the efficiency considerations of the emulation process; stack-machine architectures, sophisticated arithmetic processors, and "wide-word" machines are, for example, well within this range.

4.2 MAJOR RESOURCES AND ORGANIZATION

4.2.1 MAJOR BUSING STRUCTURE AND LOCAL STORE

The major hardware units of the QM-1 -- stores, Arithmetic-Logic Unit, shifter, register banks -- can each process or store 18 bits of data in parallel, and are connected by a system of twelve 18-bit-wide data paths (buses). The central major unit, Local Store, is a terminus for all twelve buses; the other end of each bus is connected to some other major unit. This structure is shown in Figure 4.2.1A.

QM-1 MAJOR BUSING STRUCTURE (18 BIT WIDE ARCHITECTURE) Figure 4.2.1A

```
                            --------------   ------------
  ------------------------->I MAIN STORE I-->I RMI UNIT I->---
  I                         --------------   ------------    I
  I                                                          I
  I  .....        -------   --------------                   I
  I  .AIL.------->I     I   I   SHIFTER  I           .....   I
  I  ..... .....  I ALU I-->I            I------->---.ADD.   I
  I     I  .AIR.->I     I   I EXTENSION  I           .....   I
  I     I  .....  -------   --------------             I     I
  I     I     I                                        I     I
  I     I     I             --------------             I     I
  I     I     I   .....     I            I      .....  I     I
  I     I     I   .SID.---->I   SHIFTER  I-->---.SOD.  I     I
  I     I     I   .....     I            I      .....  I     I
  I     I     I     I       --------------        I    I     I
  I     I     I     I                             I    I     I
  I     I     I     I     --------------------    I    I     I
  I     I     I     ----<-I RO I///////I R31 I--<--    I     I
  I     I     -----<------I  . I///////I  .  I         I     I
  I     -------<----------I  . I///////I  .  I-<--------     I
.....                     I  . I LOCAL I  .  I              .....
.MIX.--------<------------I  . I STORE I  .  I<-------------.MOD.
.....   -------<----------I  . I///////I  .  I              .....
  I     I     -----<------I  . I///////I  .  I-<--------      I
  I     I     I     ----<-I RO I///////I R31 I--<--    I      I
  I     I     I     I     --------------------    I    I      I
  I     I     I     I                             I    I      I
  I     I     I     I                             I    I      I
```

```
     I     I     I   .....      --------------       I     I     I
     I     I     I   .CIA.----->I            I     .....   I     I
     I     I     I   .....      I  CONTROL   I-->--.COD.   I     I
     I     I   .....            I   STORE    I     .....   I     I
     I     I   .CID.----------->I            I           I     I
     I     I   .....            --------------             I     I
     I   .....                                           .....   I
     I   .EID.              -----------------------      .EOD.   I
     I   .....              I E31 I//////////I EO I      .....   I
    /I\    I                I  .  I//////////I  . I        I     I
     I     I                I  .  I EXTERNALI   . I        I     I
     I     I  .....         I  .  I   REG'S I   . I  ..... I     I
     I     ---.EIA.--------->I  .  I//////////I  . I->-.EOA.--     I
     I        .....         I  .  I//////////I  . I  .....         V
     I                      I  .  I//////////I  . I              I
     I                      I E31 I//////////I EO I              I
     I                      -----------------------              I
     I                                     I  /I\                I
     I                                     I   I                 I
     ----------------<----------------------   --------------------
```

Associated with each bus is a direction of data flow and, in general, a distinct nanoprimitive control for the gating (transmission) of data. Since the buses and their controls are physically independent, they may be exercised in parallel, allowing a maximum of twelve program-controlled 18-bit bus transfers to occur simultaneously. Each bus bears a three-letter label structured as follows:

The first letter codes the major unit which the bus connects to Local Store:

M - Main Store
C - Control Store
A - Arithmetic-Logic Unit (and high-order half of shifter output)
S - Shifter (low order 18 bits only)
E - External Store

The second letter defines the direction of data flow:

I - Input (to the named unit from Local Store)

O - Output (from the named unit to Local Store)

The third letter provides further descriptive information:

A - Address
D - Data
L - Left
R - Right
X - "Multiplex" (used only for MIX which is shared for Main Store Address and data)

Thus the twelve buses are labeled:

MIX - Main Store Input - Address/Data	MOD - Main Store Output Data
CIA - Control Store Input - Address	
CID - Control Store Input - Data	COD - Control Store Output Data
AIL - ALU Input - Left	
AIR - ALU Input - Right	AOD - ALU Output Data
SID - Shifter Input Data	SOD - Shifter Output Data
EID - External Store Input Data	EOD - External Store Output Data

Explanation of the EOA and EIA labels in the diagram is deferred to Section 4.2.5.

Much of the programmer's attention involves the interaction of Local Store with the other major units, via the busing structure. Local Store is a bank of 32 18-bit registers, logically uniform with respect to busing. EACH BUS IS INDEPENDENTLY CONNECTABLE, UNDER PROGRAM CONTROL, TO ANY LOCAL STORE REGISTER. Connecting a bus to a register ("setting a bus control") is a primitive operation for the nanoprogrammer. Once a bus control has been set, the bus remains connected to the register until the nanoprogram changes that bus control. There is no restriction on the number or identity of buses that may be connected to any (one) Local Store register at the same time, although each bus is connected to one and only one register at any given time. Once a word appears on a bus, however, it remains available until some specific action changes the bus source. (Thus, for example, it is possible to gate the contents of a Control Store location into several Local Store registers by successively changing the COD bus control and executing the appropriate GATE nanoprimitive.)

If the data on two or more buses are gated into the same Local Store register simultaneously, the logical "OR" of the values appears in the register.

A convenient model of the busing structure represents each bus control as a "rotary switch" attached to a data path; the position of the switch as last set connects the path to one of the 32 Local Store registers. A "GATE" nanoprimitive activates data transmission on any path into Local Store.

Since the nanoprogrammer will typically use many of the Local Store registers to support the functions of some higher level emulated machine (accumulators, location counters, memory address registers, stack pointers, general-purpose registers, etc.), the bus controls effectively allow the resource organization of that machine to be dynamically redefined to best fit the current task.

4.2.2 LOCAL STORE SPECIAL FEATURES

The Local Store registers are labeled R0 through R31. In addition to their standard properties as members of Local Store, certain of the Local Store registers possess special capabilities as illustrated in Figure 4.2.2A.

```
SPECIAL FEATURES OF LOCAL STORE          -------------------------
                                         I                       I
                                         V                       I
                          ==============================         I
                      R00 I                            I         I
                      R01 I                            I         I
                          I                            I         I
            ------------------------------             I         I
            I             I              I             I         I
      =============       I              V             I         I
      I INCREMENT I       I============================I         I
      I    MPC    I   R24 I  FOUR REGISTERS AVAILABLE  I         I
      I  FEATURE  I   R25 I   WITH SPECIAL INCREMENT   I    ===========
      =============   R26 I FEATURE TO FACILITATE USE  I    I  INDEX  I
            ^         R27 I AS MICRO PROGRAM COUNTERS  I    I   ALU   I
            I             I============================I    I FEATURE I
            I             I              I             I    ===========
            ------------------------------             I         ^
                          I                            I         I
                          I============================I         I
                 ---> R31 I MICRO INSTRUCTION REGISTER I------   I
                 I        ==============================     I   I
                 I                       I                   I   I
                 I                       -------------------------
                 I                                           I
                 I        ==============================     I
Figure 4.2.2A    ---------I SIX BIT CONTROL STRUCTURE  I<-----
                          ==============================
```

4.2.2.1 MICRO INSTRUCTION REGISTER

The most important special facility in Local Store involves R31. This is the only Local Store register that is dedicated to a specific purpose. R31 is the Micro Instruction Register. When a Control Store word is executed as a microinstruction, it can readily be gated into R31 so that the nanoprogrammer may conveniently make use of the parameter information in the word (the micro-opcode is automatically cleared to zeros). Thus, R31 serves as the Micro Instruction Register (MIR).

To allow microinstruction parameters access to the QM-1 six-bit control structures (to be presented in Section 4.3), R31 is partitioned into three 6-bit fields: C,A,B (high to low order). Hence R31 also serves the special function of interfacing the QM-1 18-bit and six-bit architectures (see Section 5.3.4).

4.2.2.2 MICRO PROGRAM COUNTERS

Still a different special capability applies to four other registers in Local Store; R24, R25, R26 and R27. An Increment MPC feature is provided to facilitate the use of any of these registers as a "Micro-Program Counter" (location counter for microprograms executing out of Control Store). Controls exist for directly incrementing any of these registers by one of the following values:

+1
+2
"B" Field of R31 (six bits sign-extended; 2's complement)
"AB" Field of R31 (low order 11 bits only; sign extended)

Other elements of the MPC facility are discussed in sections 4.2.4, 4.5.3, and 5.6.5.

4.2.2.3 INDEX ALU FEATURE

An "Index ALU" capability is available for all Local Store registers other than the four MPC registers. Arithmetic operations may be performed on the contents of these registers directly using one of several quantities (in 2's complement form) without routing through the Arithmetic-Logic Unit. Selection of Index source is made from the following list:

One of 12 External Store registers
Data on the COD bus
Data on the MOD bus

Further detail on the Index Alu Feature is given in Section 5.6.4

4.2.3 ALU AND SHIFTER

4.2.3.1 ALU

The Arithmetic-Logic Unit can be controlled to perform all of the 16 logical (Boolean) operations, as well as certain arithmetic operations (including addition and subtraction), upon the two 18-bit operands present on the AIL and AIR buses. (The carry-in value for arithmetic functions is supplied by the CIH bit; see section 4.2.3.4). The 18-bit result proceeds through the Shifter Extension to the ADD bus, where it is available for gating into a Local Store register upon execution of the nanoprimitive "GATE ALU". The ALU may be used to do 2's complement, 1's complement, or unsigned arithmetic. (2's complement arithmetic is most consistent with other CPU mechanisms).

A 16 BIT MODE permits the inputs to be sign extended from 16 to 18 bits so that the operation of the ALU need not be changed when dealing with 16 bit data values.

A DECIMAL control facilitates decimal arithmetic by generating a "decimal correction word" on the SOD bus while binary functions are performed in the ALU. If the ALU propagates a carry out of a four-bit group (counting from the low-order end), "0000" is forced onto the corresponding group on the SOD bus. If no carry is propagated, "0110" is forced. The high-order two bits of SOD are forced to zeros. When the DECIMAL control is active, the Shifter Extension is automatically bypassed. The shifter input is also blocked, and has no effect on the correction word value.

ALU functions include PASS LEFT, for transferring the value on the AIL bus directly to the Shifter Extension without incurring ALU propagation delay.

4.2.3.2 SINGLE SHIFTS

The Shifter can be functioned to perform a large number of different shift operations upon the data present on the SID bus. The result is placed on the SOD bus, where it is available for gating into a Local Store register upon execution of the nanoprimitive "GATE SH". When no shift operation is specified, the Shifter functions as a direct connection from the SID bus to the SOD bus, providing a convenient route for transfers between Local Store registers.

Shift operations as described above, involving only the Shifter and the SID and SOD buses, are known as "single-length" shifts. The following types of single-length shifts can be specified:

LEFT AND RIGHT LOGICAL:
zeros inserted at one end, bits shifted off the other end.

RIGHT ARITHMETIC:
sign bit (high-order bit) extended (copied) rightward, bits shifted off right end.

LEFT AND RIGHT CIRCULAR:
rotations of the 18-bit quantity.

For each type of shift operation, shifts of any (meaningful) number of positions are performed in parallel -- i.e., as a single hardware operation. Hence single-length shifts of 0 through 18 positions can be performed directly.

Note: When single-length shifts are specified, the Shifter Extension functions as a direct connection from the ALU output to the AOD bus.

4.2.3.3 DOUBLE SHIFTS

When a double-length shift operation is specified, the Shifter Extension joins the Shifter in treating the ALU output and the value on the SID bus as the high-order and low-order halves, respectively, of a 36-bit quantity. The high-order and low-order halves of the shifted result appear on the AOD and SOD buses, respectively. In some types of double-length shifting, a 37th bit, involved in the carry function, is also used. When included, it is placed to the left of the Shifter Extension. (Carry will be further treated later.) The following types of double-length shifts can be specified:

LEFT and RIGHT LOGICAL:
: Zeros inserted at one end of a 37-bit quantity (carry included), bits shifted off other end.

RIGHT ARITHMETIC:
: Sign bit of 36-bit quantity (high-order bit of ALU result) extended (copied) rightward; bits shifted off right end of Shifter. Carry is not involved.

LEFT ARITHMETIC:
: Same as LEFT LOGICAL, except that this operation can set the Overflow condition (to be discussed), whereas LEFT LOGICAL does not set Overflow.

LEFT and RIGHT CIRCULAR:
: Rotations of the 36-bit quantity (carry not involved).

Double-length shifts of any number of positions (0 through 37) are also performed in parallel.

Note: When double length shifts are specified, the AOD bus contains the high order portion of the shifted quantity for as long as the double-shift is in control and the inputs are stable.

4.2.3.4 CARRY CONTROL

Two flip-flops are involved in carry functions within the ALU-shifting complex: the "CARRY-IN HOLD" (CIH) and the "CARRY-OUT HOLD" (COH).

The 37th bit position involved in double logical and arithmetic shifts (section 4.2.3.3) is known as the "SH END" position; it is logically located at the high-order (left) end of the double-length shift unit.

Two other elements are required in the model to be explained below:

a) Two independently programmable controls, "LEFT CTL" and "RIGHT CTL"
b) The following mutually exclusive nanoprimitive operations:

SET CIH	CLEAR CIH	SH TO COH
SET COH	CLEAR COH	ALU TO COH
		ALU TO BOTH (COH AND CIH)

Figure 4.2.3.4A aids in understanding the ALU-shift-carry system.

The output of CIH is permanently enabled as ALU carry-in, and has no other function.

CIH can be loaded from one of two sources:
a) direct program load: "SET CIH", "CLEAR CIH".
b) ALU carry-out; effected by "ALU TO BOTH".

The output of COH is permanently enabled to serve the following functions:
a) sole input to the SH END bit position.
b) one of two inputs to the "LEFT CONTROL SWITCH", to be explained.
c) sole source of the "carry test" value, one of the "local conditions" that can be tested in a nanoprogram.

COH can be loaded from one of three sources:
a) direct nanoprogram load: "SET COH", "CLEAR COH".
b) ALU carry-out; effected by "ALU TO COH" or "ALU TO BOTH".
c) output of the "RIGHT CONTROL SWITCH", to be explained.

The output of the LEFT CONTROL SWITCH is the low-order bit position of the SOD bus. In its normal state, this switch connects the SOD-low-bit to the low-order bit position of the SHIFTER output. When the "LEFT CTL" is active, however, the latter connection is broken, and SOD-low-bit is instead taken from the output of COH.

The output of the RIGHT CONTROL SWITCH serves the sole function of providing a source for loading COH (by "SH TO COH"). In its normal state, this switch loads COH from the output of the SH END bit position. When the "RIGHT CTL" is active, however, this connection is broken, and this switch instead loads COH from the low-order bit of the SID bus.

Figure 4.2.3.4A

```
       /I\ Local                                        /I\  SOD
        I  Carry                                         I   bit
        I  Test                                         /I\   0
        I                        LEFT CONTROL SWITCH       *
        I                      (normal setting shown)       *
        I                                            /I\   /I\
        I                                             I     I
        I----------------------------------------------     I
        I                                                   I
        I                                                   I
        I  RIGHT CONTROL SWITCH                             I
        I (normal setting shown)                            I
        I        <<<-----------------------------------------------
        I  <<<                                              I     I
        I  I  ***<<<I                                       I     I
        I  I        I                                       I     I
        I  I        I                                       I     I
        I  I     ===============================================  I
        I  I     I SH  I                   I               bit I  I
        I  I     I END I SHIFTER EXTENSION I     SHIFTER    0  I  I
        I  I     I BIT I                   I              (SLB)I  I
        I  I     ===============================================  I
        I  I       /I\                                     /I\    I
        I  I        I                                   SID I---->I
        I--I---------                                       I
        I  V
     =======           =====================      =======
     I COH I<----------I     A L U         I<-----I CIH I
     =======           =====================      =======
             Carry Out                     Carry In
```

4.2.3.5 TEST CONDITIONS

Including CARRY, there are six "local conditions", generated by ALU and/or shifting operations, which can be tested by nanoprimitives. They are:

CARRY (C) Output of COH, as discussed in Section 4.2.3.4.

SIGN (S) The high-order bit on the ADD bus.

RESULT (R) Normally the Logical OR of the low-order 17 bits on the ADD bus. However, when either of the special carry controls, "RIGHT CTL" or "LEFT CTL", are set, "RESULT" is the Logical OR of the low order 17 bits on the ADD bus and all 18 bits on the SOD bus. Thus an absolute zero value, either 18 or 36 bits may be tested with the condition of both S and R equal 0.

OVERFLOW (O) This condition is the logical OR of shifting overflow and ALU overflow. Shifting overflow arises only in double left arithmetic shifts, and is defined to arise if and only if a serial (bit-by-bit) shift of the same number of positions would, at any time, change the value of the high-order (sign) bit of ADD. ALU overflow arises (see Section 5.6.2), if and only if the bit-carry signals propagated into the sign and carry-out positions are of opposite values.

SHIFTER HIGH BIT (SHB) The high-order bit on the SOD bus.

SHIFTER LOW BIT (SLB) The low-order bit on the SOD bus.

Since it is highly desirable to have a convenient method of preserving condition bits, a location is provided in the CPU for a copy of each of the six local conditions. The nanoprogrammer can set a control such that when GATE ALU is executed, the four local conditions C,S,R, and O are automatically copied into their corresponding "global condition" bits, and (under separate control) when GATE SH is executed, the two local conditions SHB and SLB are copied into their global counterparts. Then, independent nanoprimitive tests can be made upon these "global conditions".

4.2.3.6 SIXTEEN-BIT MODE

A special CPU feature is included to facilitate manipulation of byte-oriented data. A "16-BIT MODE" control can be set by the nanoprogrammer, with the following effects:

a) The local conditions S, R, and SHB are redefined to function as if the Shifter and Shifter Extension were each 16 bits wide, with the virtual units mapped onto the low-order 16 bits of the 18-bit units (i.e., the S-test is taken from ADD bit 15 instead of ADD bit 17 -- using 0-origin numbering; etc.). The "double width" R test is based on the lower 33 bits of the concatenated ADD and SDD buses.

b) The RIGHT CONTROL SWITCH selects bit 16 of the ADD bus instead of the output of the SH END bit to load COH.

c) The ALU Overflow condition and ALU carry-out are redefined to function as if the ALU were 16 bits wide, with the virtual ALU mapped onto the low-order 16 bits of the 18-bit ALU. Note: shifting Overflow is not redefined.

d) The AIL and AIR buses automatically copy the 3rd-highest-order bit (i.e., the sign bit of a 16-bit word) into the two high-order positions; thus arithmetic in 16-bit mode also generates correct 18-bit results, for later use in 18-bit mode if desired. Important: the "PASS LEFT" ALU function, which routes the contents of AIL directly around the ALU, also bypasses this sign-extension mechanism.

4.2.4 CONTROL STORE

Control Store is a fully readable/writable general-purpose 18-bit wide store, implemented in semiconductor memory. It is available in blocks of 1K words, up to a maximum of 16K words.

The nanoprimitives "READ CS", "WRITE CS", and "GATE CS" are provided to access Control Store. The READ CS and WRITE CS nanoprimitives are accompanied by a field which selects the source of the address in Control Store at which a word is to be accessed. Sources of CS addresses are as follow:

CIA: The value on the CIA bus; for general-purpose data access.
COD: The value on the COD bus; for convenient indirect access.
MPC, MPC+1, MPC+2, MPC+B, MPC+AB (low-order 11 bits of R31): (Increments sign-extended, 2's complement.) For microinstruction sequencing and branching, and for reading microinstruction parameter lists; microinstruction execution is discussed in section 4.5.3. Selection of which of the four MPC's is to be used is made by a mechanism similar to a bus control (see section 4.3.2.3)
INDEX: The (18-bit) value taken from the INDEX ALU Output bus (see section 5.4.2.1)

When a word has been read out of Control Store, it appears on the COD bus, available for gating into a Local Store register by execution of the nanoprimitive GATE CS. Once established, a COD value remains until changed by the next READ CS or WRITE CS operation.

Writing a word into Control Store is accomplished by placing the datum on the CID bus, and then executing the nanoprimitive WRITE CS with the appropriate CS address selected. The newly written value then appears on COD.

If READ CS and WRITE CS are executed simultaneously, READ CS is ignored.

Execution of READ CS from (or WRITE CS to) a nonexistent location generates zeros on the COD bus; nothing in Control Store is changed in either case. NOTE: Negative addresses(bit 17 on) will execute READ CS from READ - ONLY Control Store(i.e. ROCS; see section 4.8)

4.2.5 EXTERNAL STORE

External Store is a bank of 32 registers, partitioned into several groups to support specific functions: External ports, Index registers, Main Store addressing facilities, and interrupt control.

Although each type of ES register is associated with special hardware facilities to implement its specific function, all 32 ES registers are uniformly accessible by the nanoprogram via the EOD and EID buses. To provide this accessibility, two additional bus controls are associated with External Store transfers, as follows:

While the destination end of the EOD bus is connected to one of the 32 Local Store registers by the EOD bus control, the source end of the same bus is connected to one of the 32 External Store registers by a different bus control, labeled EOA. The transfer from ES to LS is executed by the nanoprimitive "GATE ES". Similarly, the External Store connection of the EID bus is selected by the EIA bus control. The transfer from LS to ES is executed by the nanoprimitive "LOAD ES".

Functions of External Store registers are as follows:

E0 through E7: Eight Port Registers available for interfacing the QM-1 to its environment. These registers are directly connectable to Main Store. (The QM-1 external interface is discussed in section 4.6.)

E8 through E19: Twelve Index ALU Operand sources. These include eight registers for general use and four registers from the groups below.

E16,E17: BASE ADDRESS register and FIELD LENGTH register associated with Main Store addressing (discussed in section 4.2.6.3). Inclusion of these machine-control functions in E16 and E17 is a QM-1 OPTION; if such functions are not included, E16 and E17 are scratch registers.

E20,E21: ALTERNATE BASE Address and FIELD Length registers.

E18 through E31: The remaining twelve registers are associated with Interrupt control (see section 4.5.2.4). These registers include:
E18 and E19: Interrupt Enable Bits.
E22 --- E31: Interrupt Address Fields.

The overall layout of External Store is shown in Figure 4.2.5A

```
         I-----------------------------------------------------I  ---------
E0       I                                                     I        R
         I-----------------------------------------------------I        E
E1       I                                                     I    P   G
         I-----------------------------------------------------I    O   I
        ---                                                   ---   R   S
         I-----------------------------------------------------I    T   T
E6       I                                                     I        E
         I-----------------------------------------------------I        R
E7       I                                                     I        S
         I-----------------------------------------------------I  ---------------
E8       I                                                     I              ^
         I-----------------------------------------------------I              I
        ---                                                   ---        INDEX  ALU
         I-----------------------------------------------------I          OPERANDS
E15      I                                                     I              I
         I-----------------------------------------------------I  ---------   I
E16      I           MAIN STORE BASE ADDRESS REGISTER          I   MAIN       I
         I-----------------------------------------------------I  STORE       I
E17      I           MAIN STORE FIELD LENGTH REGISTER          I  CONTROLS    I
         I-----------------------------------------------------I  ---------   I
E18      I PROGRAM/02 03 04/05 06 07/08 09 10/11 12 13/14 15 16I  INTERRUPT   I
         I  CHECK /--------------------------------------------I   ENABLE     I
E19      I  MASKS /17 18 19/20 21 22/23 24 25/26 27 28/29 30 31I  CONTROLS    V
         I-----------------------------------------------------I  --------------
E20      I            ALTERNARE BASE ADDRESS REGISTER          I  ALTERNATE
         I-----------------------------------------------------I  MAIN STORE
E21      I            ALTERNATE FIELD LENGTH REGISTER          I   CONTROLS
         I-----------------------------------------------------I  ---------
E22      I         02        /        03        /        04    I    I  A
         I-----------------------------------------------------I    N  D
E23                05        /        06        /        07    I    T  D
         I-----------------------------------------------------I    E  R
        ---                                                    I    R  E
         I-----------------------------------------------------I    R  S
E30                26        /        27        /        28    I    U  S
         I-----------------------------------------------------I    P  E
E31                29        /        30        /        31    I    T  S
         I-----------------------------------------------------I  ---------
Bit        17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
```

LAYOUT OF EXTERNAL STORE   Figure 4.2.5A

4.2.6 MAIN STORE

4.2.6.1 GENERAL

Main Store is a general-purpose 18-bit-wide core storage, available in blocks of 8K words up to 256K maximum (16K words minimum). Full cycle time is 800 nanoseconds; since lower-level control operations occur an order of magnitude faster, Main Store is well suited to contain programs of virtual (emulated) machines whose instructions require a moderately complicated interpretation at lower levels.

For convenience and efficiency in Input/Output processing, the two buses associated with Main Store (MIX and MOD) may be connected not only to any of the 32 Local Store registers, but also to any of the 8 Port Registers in External Store. Thus for the MIX and MOD bus controls only, the Port Registers are treated as extensions to Local Store; they are designated as R32 through R39 when used in this way.

4.2.6.2 MS OPERATIONS

To initiate a full (non-destructive) read operation in Main Store, the nanoprogrammer first determines that Main Store is not busy ("MS BUSY" is one of the "special conditions" available for nanoprogram testing), and then simultaneously executes the two nanoprimitives "MSGO" and "MSRS" (for "Main Store Restart"). Main Store accesses the location addressed by the value on the MIX bus, as modified by addressing facilities which are discussed in the next section.

When the accessed word is available, another special test condition, "MS DATA INVALID", becomes false, and the nanoprogram can gate out the word through the MOD bus by executing the nanoprimitive "GATE MS"; access time is 640 nanoseconds.

The same two nanoprimitives, MSGO and MSRS, are used to control other operations of Main Store, as follows;

When MSGO is executed without MSRS, Main Store begins the first half-cycle ("extraction part") of a split-cycle operation. As in the case of a full-read operation, the address is taken from the MIX bus. Main Store will remain BUSY until the completion of the second half-cycle ("insertion part"). In the split-cycle mode of operation, however, the latter must be explicitly invoked by the nanoprogram execution of MSRS; the data word to be inserted (written) is taken from the MIX bus at the time MSRS is executed. This mode of operation may be used in two ways:

To perform a Read/Modify/Write sequence, the nanoprogrammer initiates split-cycle operation as described above and then, when MS DATA INVALID becomes false, gates out the extracted word (GATE MS) for modification (for example indexing) by CPU facilities. When the modified word is ready for insertion back into its MS location, it is placed on the MIX bus and MSRS is executed. Since MS BUSY will become false at the completion of the second half-cycle, this operation can offer significant time savings over the alternative full-read, data manipulation, full-write sequence, especially if the modification period is relatively short.

Alternatively, to perform a "full write", the nanoprogrammer initiates split cycle operation as described above and then AT ANY TIME THEREAFTER -- including immediately after MSGO -- places the word to be inserted on the MIX bus and executes MSRS. If MSRS is received during the first half-cycle of split-cycle operation, Main Store will latch the data-in word from the MIX bus and "remember" to initiate the second half-cycle as soon as possible.

A distinct advantage to lessening the distinction between the full-write and Read/Modify/Write functions is that the nanoprogram can initiate a Main Store operation without making a commitment to either of the two functions; if the nanoprogram subsequently decides that the operation is to be a full-write, no time loss is incurred if the decision is made before the end of the first half-cycle. (In fact, the full-read function may also be achieved in the split-cycle mode, although with a slight degradation in cycle time due to routing delays.) This facility is thus useful in implementing certain Main Store modification look-ahead schemes.

Notes:

1. Main Store ignores any MSGO signal received when MS BUSY is true. (See section 5.4.3.)

2. Main Store ignores any MSRS signal received when either:
   a) MS is not BUSY; or
   b) MS is BUSY in full read mode; or
   c) the second half-cycle in split-cycle mode is already in progress. (See section 5.4.3.)

3. When either mode of operation is initiated:
   a) MS BUSY becomes true;
   b) MS DATA INVALID becomes true; and
   c) MOD is cleared to zeros.

4. When the second half-cycle of split-cycle mode is initiated, MOD takes on the value of the word being inserted. Note, however, that MS DATA INVALID is set "true".

4.2.6.3 MS ADDRESSING AND PROTECTION (QM-1 OPTION)

A QM-1 OPTION is the use of External Store register 16 as the BASE register and External Store register 17 as the FIELD LENGTH register associated with the base-addressing, write-protection, and address-alarm facilities to be described below. If this option is not present, then:

a) E16 and E17 are available for scratch use; and
b) the facilities described below operate as if the BASE register permanently contained the value zero and the FIELD LENGTH register permanently contained the value 2**18-1.

Whenever Main Store uses the value on the MIX bus as an address, that value is treated as a displacement; it is added to the contents of the BASE register to yield the true (absolute) address to be accessed.

An MS ADDRESS VIOLATION Program Check is generated in either of the following two cases:

a) When an MS access of any kind uses an absolute address which falls outside the allowed range defined by the BASE and FIELD LENGTH registers. The lowest physical address in the allowed range is the value of the BASE register; the number of words (consecutive locations) in the allowed range is given by the contents of the FIELD LENGTH register(i.e. 0 < the number of words accessable < c(FIELD LENGTH) +1). Wraparound is disallowed.
b) When an MS access of any kind addresses a location which is not physically present in Main Store.

For the convenience of programs used as "privileged" or "system" routines, a nanoprimitive control ("DIRECT MS ACCESS") can momentarily force the effective value of the BASE REGISTER to zero and the effective value of the FIELD LENGTH REGISTER to 2**18-1 (E16 and E17 themselves do not change value).

In addition to generating the Program Check, detection of MS address violation sets MOD to all ones and leaves the contents of the memory unchanged.

In all modes of Main Store operation, a Program Check is generated in the case of failure of a parity test automatically made upon the extracted word.

4.2.6.4 RMI UNIT (QM-1 OPTION; FURTHER SPECIFICATIONS TO BE ANNOUNCED)

If desired, the contents of the MOD bus can be routed through the RMI unit before being gated into a Local Store register or a Channel Register.

The RMI unit - Rotate, Mask, and Index - is a very general data-transformation device with special application in extracting fields and decoding information when emulating a "Main Store Machine". The operation of the RMI unit passes a word through three successive stages of transformation:

a) The initial value undergoes a right circular shift by the number of positions specified in a ROTATE parameter.
b) The result of this operation is logically ANDed with an 18-bit MASK parameter.
c) The result of this operation is added (2's complement) to an 18-bit INDEX parameter to yield the final result.

There are three sets of such parameters. They are loaded with three separate AUX Actions, and the data is taken form the COD bus(see section 5.8.2). Selection of one of the three parameter sets is associated with the GATE MS nanoprimitive; a fourth option is to bypass the RMI unit.

Notes:

1. Since the MOD source value remains stable until changed by a Main Store operation, the same word may be taken through a succession of different RMI transformations (and also routed to different destinations) without re-cycling Main Store.
2. The RMI parameters may be changed as often as desired by the nanoprogrammer.

4.3 SIX-BIT CONTROL STRUCTURE

4.3.1 GENERAL

The large number of hardware resources in the QM-1 and the flexibility with which they may be used require a large variety of control information, dynamically changeable during the execution of user programs.

Rather than having all such information placed in a store from which instructions are executed, the concept of "residual control" is implemented. Registers are provided in the machine for holding this hardware-controlling information. These registers can be loaded at the explicit command of an executing program; their contents will remain in control of their assigned hardware functions until reloaded (hence "residual control"). In this manual, the terms "residual control" and "nanoprimitive control" are used with mutually exclusive definitions.

Residual control functions in the QM-1 are maintained in a bank of six-bit registers known as F-store. A complete system of nanoprimitive controls and six-bit data paths exists for transferring quantities between F-registers and a set of six-bit source and destination fields elsewhere in the machine, and for manipulating these data.

The six-bit source and destination fields (from the point of view of F-store) are collectively known as Auxiliary (AUX) fields, although some are control registers in their own right.

The same rules of simultaneous busing apply to both 18-bit and six-bit transfers, such that if two or more AUX Fields are gated into the same F Register simultaneously, the logical "OR" of those source values appears in the F Register. Two or more F Registers bussed to the same AUX Field, however, produces the logical AND of the values.

4.3.2 F-STORE

4.3.2.1 GENERAL

The 32 six-bit registers in F-store, numbered F0 through F31, are all uniformly accessible for the purpose of loading from six-bit source fields and reading into six-bit destination fields. Execution of such six-bit transfers and the associated addressing in F-store (as well as selection of source and destination fields) are accomplished entirely by nanoprimitive control.

Similarly, nanoprimitive controls may be applied uniformly to any register in F-Store to INCREMENT (by one) or DECREMENT (by one) the contents of that register (modulo 64).

It is convenient, however, to approach F=Store as partitioned into three groups, by function: bus control F's, special F's, and G's. This is shown in Figure 4.3.2.A.

```
          F STORE                                         AUX FIELDS
-------------------------                        -----------------------
I FMIX                    I                      I THREE FIELDS IN R31 I
I FMOD                    I   /                  I       C, A, B       I
I FCIA                    I  /----------         I---------------------I
I ----     BUS            I /      IN            I K FIELDS IN THE     I
I ----     CONTROLS       I \ TRANSFERS          I EXECUTING NANOWORD  I
I FEOD                    I  \----------         I                     I
I FEIA                    I   \                  I ALU CONTROL-KALC    I
I FEOA                    I                      I SHIFT CONTROL-KSHC  I
I-------------------------                       I SHIFT AMOUNT-KSHA   I
I ----                    I                      I TEST MASKS-KS,KX,KT I
I FMPC     SPECIAL        I                      I CONSTANTS-KA,KB     I
I ----     CONTROLS       I           \          I---------------------I
I FIPH                    I -----------\         I    MISC. SOURCES    I
I-------------------------I     OUT     \        I---------------------I
I G0       CONSTANTS,     I   TRANSFERS /        I      G0 - G11       I
I ----     BACK-UP        I -----------/         I  (AVAILABLE ONLY    I
I G11      CONTROLS       I           /          I    AS SOURCE AUX)   I
-------------------------                        -----------------------
```

SIX BIT ARCHITECTURE FIGURE 4.3.2A

4.3.2.2 BUS CONTROL F's

The first fourteen F Registers are the bus controls(see section 4.2.1). They are symbolically referenced in association with their bus names (FMOD, FAOD, etc.), with the addition of FEIA and FEOA. The contents of these registers are interpreted in one of three ways to achieve bus control, depending on the nature of the associated bus.

The contents of an F-register associated with the DESTINATION end of a bus (with the exception of FMOD) are used modulo 32 to address (connect) a Local Store register (FSOD, FAOD, FEOD, FCOD) or, in the case of FEIA, an External Store register. (The high-order bit is ignored for bus control purposes, although it is physically present in the F-register as loaded.)

The contents of an F-register associated with the SOURCE end of a bus (with the exception of FMIX) are used modulo 64 to address (connect) a Local Store register (FSID, FAIL, FAIR, FEID, FCIA, FCID). If the address is greater than 31 (i.e., if the high-order bit is set), the bus is connected to a permanent source of all ones, rather than to a Local Store register. In the case of FEOA greater than 31, the EOD bus is connected to a source of all zeros rather than to an External Store register. (i.e. If FEIA > 31 then LOAD ES wraps around the E Registers; if FEOA > 31 GATE ES sends zeroes to LS.)

Since MIX and MOD have an addressing range beyond 32 (see section 4.2.6.1), special rules are used in interpreting the bus control functions of FMIX and FMOD; these F-registers are used modulo 64, with the eight Port Registers (E0 through E7 -- see section 4.2.5) treated as contiguous extensions to Local Store for this purpose. Since the MIX and MOD buses may not be connected to an External Store Register beyond the Port Registers, the following rules apply:

1. When FMIX contains a value greater than 39 (corresponding to E7, the last Port Register), the MIX bus takes the value of all ones.
2. When FMOD contains a value greater than 39, GATE MS is a null operation.

4.3.2.3 SPECIAL F's

The next six F-registers serve special control functions, some of which have been mentioned previously.

FACT: (Auxillary ACTion) FACT is used as a source value to specify a variety of special action commands. These are described in Section 5.8.2.

FUSR: (Control Store USeR partition number) Bits 0 thru 3 identify the Control Store partition currently accessible by the CPU. This function is meaningful only in those systems utilizing the Control Store Address Translation option(see Appendix B). Bits 4 and 5 are ignored. When the CS Address Translation is not in use FUSR is a general scratchpad 6 bit register.

FMPC: The contents of FMPC, modulo 4, select one of the four Micro Program Counters in Local Store to be used for MPC operations (see section 4.2.2). The selection is according to:

FMPC (mod 4)	MPC
0	R24
1	R25
2	R26
3	R27

FIDX: (InDeX) FIDX has one main function and three auxilliary ones:

```
 ---------------------------------------------------
 / 16 MODE / SUPR ST / RONS /   NS  PAGE  INDEX   /
 ---------------------------------------------------
Bit   5         4        3        2     1     0
```

Bit 5: 16-BIT-MODE control; "1" for 16-bit mode, "0" for 18.

Bit 4: Supervisor instruction State; "1" allows entry to supervisory (restricted) nanoprograms (section 4.5.2.2)

Bit 3: NANOSTORE Mode Control (section 4.8); "0" for normal, "1" for read-only

Bits 2, 1, 0: Nanostore Page Index, used in Nanostore addressing under Micro control. (Sections 4.5.3, 4.5.4)

FIST: (STatus) The six bits of FIST contain the "global conditions" mentioned in section 4.2.3. Since the FIST bits can be tested by nanoprimitive controls which are independent of those used to test the "local conditions", and since any F-register can be loaded with a six-bit quantity, FIST may also be used as a general-purpose bit-testing facility.

The FIST test bits are:

	SHB	C	S	R	O	SLB
Bit	5	4	3	2	1	0

FIPH: (PHantom) FIPH is a special F that gives the nanoprogrammer the ability to transfer a value from a source AUX to a destination AUX without using two T-steps as would be required when going through an F register, and without destroying the value in an F register. This is possible because FIPH is not truly a register. Having no data-storing capability, it is a direct connection between the input and output bus structures of F-store. If nanoprimitives are simultaneously executed to INput to FIPH from a six-bit source field and OUTput from FIPH into a six-bit destination field, the result is a direct transfer from the source field to the destination field. If only a load into FIPH is executed, there is no effect (except as a function code in I/O operations,section 4.6). If only a gate out of FIPH is executed, the transferred value is zero. Note: Transfers from source AUX to destination AUX via FIPH must be placed in a STRETCHed T-step.

4.3.2.4 G's

The last twelve registers in F-store are known as G-fields, or G's. The G's have no direct dedicated machine-control functions in themselves, but are used in programming systems to store back-up control information, as follows:

First, since the G's are a part of F-store, any G may be loaded from any source AUX, or read into any destination AUX. (Hence one use of G's is for temporary, or scratch, storage in six-bit programming, without inhibiting the use of machine functions.)

Second, the G's have the special property that they are also addressable as source AUX fields, and hence may be transferred directly to any register in F-Store(including G's). Therefore the G's provide space for the programmer to store control information that will subsequently be transferred into (cr exchanged with) bus control F's, special F's, and/or other control registers (i.e., certain destination AUX fields). In this sense, the G's serve the function of a "second level" of residual control.

Note: As a source, G's may be referred to as G's or as their corresponding F's, depending on the transfer, e.g. G0 = F20. As a destination they may only be F's.

4.3.3 AUX FIELDS

Available source AUX fields (which can be loaded into F-store) are:

C, A, B:	The three six-bit fields of R31, as introduced in Section 4.2.2.
KA, KB, KX, KT, KS:	Six-bit fields from the executing nanoword to be discussed below and in Section 4.5.1. KA and KB ordinarily are used for constant and/or scratch storage.
GO - G11:	The 12 G's (see Section 4.3.2.4).

The following additional source AUX's are not registers:

ALUF:	Output of six-bit ALU, to be discussed in Section 4.3.4 (QM-1 option).
IO ID:	A six-bit IDentification number associated with a device on an external port; see Sections 4.6, 5.5.2.
INCF1, DECF1, INCF2, DECF2:	Increments and decrements of F-store elements (Sections 5.5.2, 5.6.6).
SW:	Six external switches on the engineering control panel (see Figure 5.9.1A)

Available destination AUX fields (to which F-store can be output) are:

C,A,B:	See above.
KA,KB:	See above.
KALC:	A field which specifies the operation of the ALU (ALU Control).
KSHC:	A field which specifies shifting operations (Shift Control).
KSHA:	A field which specifies number of positions to shift (Shift Amount).
KS:	A six-bit mask field associated with global condition (FIST) testing.
KT:	A six-bit mask field associated with local condition testing.
KX:	A six-bit mask field associated with special condition testing.

KA, KB, KALC, KSHC, KSHA, KS, KT, and KX are all six-bit fields in the executing nanoword (see section 4.5.1).

4.3.4 ALUF (QM-1 OPTION; FURTHER SPECIFICATIONS TO BE ANNOUNCED)

A six-bit ALU, similar in characteristics to the 18-bit ALU, operates under nanoprimitive control to generate arithmetic and logical functions from two six-bit inputs.

The left and right inputs to "ALUF" are any register in F-store (selected by nanoprimitive controls).

The output of ALUF may be loaded into any register in F-store.

If the ALUF is not present, operations defined to gate its output produce an all ones(63.) value (see section 5.6.7 for further description).

4.4 TIMING

The QM-1 CPU is a synchronous device, driven by a single machine clock whose period is 80 nanoseconds.

To allow the hardware-level user intimate access to and control of QM-I hardware facilities, nanoprogram steps are executed at the machine clock rate.

To avoid ambiguity in discussing three closely related concepts, the following terms are used in this manual:

T-PERIOD:	A period of elapsed time equal to the clock period; 80 nanoseconds.
T-STEP:	An elementary event in program control; a single step of nanoprogram execution. A T-step consists of the simultaneous (parallel) execution of some number of nanoprimitive commands (nanoprimitives). A T-step generally occurs in one T-period, but for certain purposes it may be expanded (by the "STRETCH" nanoprimitive) to last for two T-periods.
T-VECTOR:	A string of bits representing a set of nanoprimitives to be executed concurrently in a single T-step. The "active T-vector" corresponds to the "current T-step". (Program control is presented in detail in section 4.5.)

When there is no danger of confusing the three concepts of time (T-PERIOD), event (T-STEP), and physical entity (T-VECTOR), the term "T" may be used; for example: "A 24-hour QM-1 working day is equivalent to more than a trillion T's." ("T" is derived from "TEE": Time, Event, Entity.)

All nanoprimitives may be classified as either "leading-edge" (LE) or "trailing-edge" (TE), according to whether the function they define takes effect at the beginning or the end, respectively, of the T-step in which they are executed. In general, the effect of trailing-edge nanoprimitives (the larger class) may be considered to occur at the end of the T-step in which they occur. For example, all nanoprimitives which transfer values into registers (18-bit or six-bit) are trailing-edge. Leading-edge nanoprimitives, on the other hand, initiate processes which have a duration of one or more T-periods; examples are READ CS, MSGO. The duration of such processes are measured from the beginning of the T-step in which their nanoprimitives are executed. The "STRETCH" nanoprimitive separates the leading edge of a T-step from the trailing edge by one extra T-period.

The difference between a T-step and a T-period is important when both leading-edge and trailing-edge nanoprimitives are programmed. For example, if READ CS (leading-edge) and GATE CS (trailing-edge) are programmed in the same T-step, and that T-step is not STRETCHed, Control Store will not generate the new value on the COD bus in time for the GATE CS. If, however, the T-step is STRETCHed, the value gated into Local Store will be that generated by the READ CS, since the time span between the leading and trailing edges of a STRETCHed T-step is two T-periods, enough for a Control Store cycle. (Timing considerations for programming Control Store and other leading-edge operations will be discussed in detail in section 5.3)

All register transfers, both 18-bit and six-bit, are controlled by trailing-edge nanoprimitives. Since these operations are synchronous, the same register effectively can be loaded and read in the same T-step ("simultaneously"), without loss of data. Given the model that register "REG" is to be gated to register "DEST" and loaded from register "SOURCE" simultaneously, then the state before the operation is that the data from SOURCE is present on the bus from SOURCE to REG, and the data in REG is present on the bus from REG to DEST; the effect of the nanoprimitives is then to latch ("clock in") the bus values into REG and DEST. Propagation delays are such that DEST will have latched its new value before the new value in REG has time to reach the bus from REG to DEST. It is quite important, however, that the new values are on the buses and ready for a repeat operation within one T-period; this fact is fundamental in nanoprogramming.

The basic timing structure of the QM-1 is derived not only from hardware considerations, but also from the design objective of being able to nanoprogram certain operations and sequences of operations efficiently. The three most important such sequences are listed below (in the symbolic program examples, the T-steps are not stretched).

1. Closed loop through Local Store. For example, let the EOD and EID buses both connect the same Local Store register and External Store register (FEOD=FEID, FEOA=FEIA), and then execute the following T-steps:

```
        Tn:     LOAD ES, GATE ES.
        Tn+1:   LOAD ES, GATE ES.
```

the result is a double exchange (final status = initial status) of the contents of the two registers.

2. Closed loop through F-store;e.g.:

```
        Tn:     AUX(x)---->F(y), F(y)---->AUX(x).
        Tn+1:   AUX(x)---->F(y), F(y)---->AUX(x).
```

The result is a double exchange (final status = initial status) of the contents of AUX(x) and F(y).

3. Bus setting immediately prior to bus use; e.g.:

```
        Tn:     AUX(x)---->FSOD.
        Tn+1:   GATE SH.
```

The Shifter output is gated into that Local Store register "pointed to" by the number contained in AUX(x) at the beginning of T-step T(n).

THE ABOVE THREE TYPES OF SEQUENCES ARE MUTUALLY CONSISTENT. For example, the third illustration could be expanded to include the T-step:

```
        Tn-1:     F(y)---->AUX(x).
```

which would set AUX(x) in time for the described sequence to occur with the Local Store register number specified by F(y).

To achieve these objectives, the machine clock signal that controls six-bit operations (the F clock) is phased differently from the signal that controls 18-bit operations (the R clock). Both are derived from the same single machine clock that controls the T-vector (the T clock). This phasing is normally transparent to the nanoprogrammer, and must be considered mainly when dealing with the interface between the six-bit architecture and the 18-bit architecture which exists in R31; the necessary programming rules are specified in detail in section 5.3.5.

4.5 NANOPROGRAM CONTROL

4.5.1 CONTROL MATRIX

The current T-step is specified by the contents of the active T-vector. This T-vector remains active for one T-period (or two, if it includes the STRETCH nanoprimitive). The active T-vector is one of four T-vectors resident in a structure known as the Control Matrix. This structure is shown in Figure 4.5.1A.

```
    CONTROL MATRIX
  (EXECUTES NANOWORD)     ------------------         ----------------------
                          I                I         II   PROGRAM CHECK   I
 ---------------------    I    ---------   I         II    (ADDRESS 0)    I
 I      K VECTOR     I    I    I       I   I         I--------------------
 I(INCLUDES AUX FLDS)I-----    I       I   ---------->II  NANOBRANCH (KN)  I----
 I-------------------I         I   N   I              II     ADDRESS       I   I
 I                   I         I   A   I              I--------------------    I
 IT1  NANOPRIMITIVE  I   /     I   N   I  ------------II INTERRUPT ADDRESS I   I
 I--  MACHINE CONTROLI  /------I   O   I / I NANOSTORE II      (2)         I   I
 I    FROM CIRCULAR  I /       I   S   I/--I ADDRESS  I--------------------    I
 IT2  ACTIVATION     I \       I   T   I\--I SELECT   I         .              I
 I--  OF T VECTORS   I  \------I   O   I \ I MECHANISM I        .              I
 I                   I   \     I   R   I  ------------I         .              V
 IT3                 I         I   E   I              I--------------------    I
 I--                 I         I       I              II INTERRUPT ADDRESS I   I
 I                   I         I       I              II      (31)         I   I
 IT4                 I         I       I              I--------------------    I
 ---------------------         ---------  LOADED FROM II NANOPROGRAM (NPC) I   I
                                      ^  ------------->II     COUNTER       I   I
 ==================   LOADED FROM  I      COD BUS     ----------------------   I
 I PROGRAM CONTROL I  -------------                    I          ^  ^          I
 I  Figure 4.5.1A  I    EOD BUS                        I SEQUENCE I  I FROM KN I
 ==================                                    -----------  ----------
```

At any given time, the Control Matrix contains 360 bits, corresponding to the 360 bits in a nanoword. This includes the four T-vectors, one of which is active (72 bits in each T-vector) -- and a 72-bit entity, the "K-vector". (The AUX registers KA, KB, KALC, KSHC, KSHA, KS, KT, and KX are all portions of the K-vector.) Since the Control Matrix bits correspond to some 360-bit word ("nanoword") in Nanostore, they may be referred to as the "active nanoword". Because of this correspondence, every word in Nanostore is logically partitioned as follows:

[K-vector] [T-vector(1)] [T-vector(2)] [T-vector(3)] [T-vector(4)]

(or, briefly: K, T1, T2, T3, T4 -- high-order to low-order)

Mechanisms are provided for selecting a nanoword, fetching that word from Nanostore, and loading it into the Control Matrix. When the nanoword is loaded into the Control Matrix, its first T-vector (T1) immediately becomes the active T-vector (and its K-vector becomes active).

The normal operation of the Control Matrix activates the four T-vectors in succession and circularly, with no loss of time between activations: T1, T2, T3, T4, T1, etc. Unless a special high-priority facility (Program Check) interrupts, this sequence continues until certain program-control nanoprimitives are executed. These nanoprimitives can be programmed to execute conditionally, so that the nanoprogrammer may create a useful loop in a single nanoword. (For example, the F ZERO test may be used; see Section 5.7.) If the programmer does not need such looping, then the sequence may of course be broken after the first activation of T4 (or earlier, if desired).

For protection against infinite looping, a Control Matrix Time-Out facility breaks the loop and generates a Program Check if the same nanoword circulates in the Control Matrix for approximately one second (more than 12 million T-periods).

The two program control nanoprimitives of immediate interest are "SKIP" and "GATE NS". Either can be executed conditionally, according to the T-vector test facilities specified in section 5.7. The bit structure in the T-vector is such that SKIP and GATE NS are mutually exclusive in the same T-step. However, another nanoprimitive "GATE NS UNCONDITIONALLY" is provided to avoid this restriction and to permit conditionally skipping T1 of the next nanoword.

SKIP, when executed, modifies Control Matrix operation so that the next T-step in succession is skipped over; activation of the succeeding T-vector is inhibited. The skipped T-step consumes one T-period of time (whether STRETCHed or not), which should be observed when leading-edge processes are programmed; the effect is the same as if the succeeding T-vector were activated, but with all its specified nanoprimitives (including STRETCH) "turned off". Note that a SKIP executed in T3 results in T1 being the next T-vector activated; a SKIP in T4 goes to T2.

GATE NS is a trailing-edge nanoprimitive which, when executed, causes the Control Matrix to be loaded with the nanoword resulting from the most recently completed Nanostore access. The successor to the T-step in which GATE NS is executed is generated by the first T-vector (T1) of the newly gated nanoword; no time is lost in the transition. The K-vector is also loaded from the K-vector of the nanoword as a result of the GATE NS.

Note: any six-bit transfers into K-vector AUX fields commanded in a T-step in which GATE NS is also executed result in undefined values, unless the "HOLD" control is on (see section 4.5.2.3).

To supply a nanoword for gating into the Control Matrix, the leading-edge nanoprimitive READ NS is executed. Nanostore completes the read operation within two T-periods (but not within one T-period); hence either of the following program examples illustrate a successful shortest-time sequence of the operations READ NS, GATE NS:

```
a)  Tn:     READ NS.  (not stretched)
    Tn+1:   GATE NS.  (not stretched)

b)  Tn:     STRETCH, READ NS, GATE NS.
```

The READ NS nanoprimitive has a secondary effect, involving the "nanobranch" facility; this is discussed in section 4.5.2.3.

4.5.2 NANOSTORE ADDRESSING

4.5.2.1 PRIORITY SELECT

When READ NS is executed, a priority-select mechanism supplies the actual nanostore address from a list of potential addresses. Each potential address value is 10 bits wide, since Nanostore may contain as many as 1024 nanowords.

Nanostore is available in 256-word blocks, and can be arranged so that any of eight possible 128-word "pages" is full, half-full, or empty. Execution of READ NS from a nonexistant location generates a zero nanoword; if such a word is loaded into the Control Matrix, no nanoprimitive operations are invoked, and a Control Matrix Time-Out Program Check eventually results.

Each source of potential nanostore address has a fixed priority relative to the other sources; furthermore, an ACTIVE/INACTIVE status is associated with each source at any given time. When invoked, the priority-select mechanism selects the address from the highest-priority source that is currently ACTIVE and supplies it to Nanostore. If the corresponding nanoword is then executed (GATE NS occurs before the next READ NS), the address source is reset to INACTIVE status. The source with lowest fixed priority is the NanoProgram Counter (NPC); this is defined as permanently ACTIVE, and can be considered a default.

The process can be modeled by a list of activation flags, each associated with a nanostore-address source, ordered by the priority of the sources; operation of the priority-select mechanism is equivalent to reading down this list, from high-priority to low-priority, until the first ACTIVE flag is encountered. The address associated with that flag is then supplied to Nanostore, and the flag is turned off (INACTIVE) upon successful use (GATE NS) of the nanoword fetched from that location.

Figure 4.5.2.1A illustrates the model and identifies the various address sources, to be discussed in the following sections.

FIGURE 4.5.2.1A

PRIORITY SELECTION OF NANOSTORE ADDRESS

```
FLAGS              SOURCE (HIGHEST PRIORITY AT TOP)



 [ ]               PROGRAM CHECK

 [ ]               NANOBRANCH

                    --                         --
 [ ]               I                             I
  .                I                             I
  .                I  INTERRUPTS (MAXIMUM 30)    I
  .                I                             I
 [ ]               I                             I
                    --                         --

 [X]               NANOPROGRAM COUNTER (NPC)
                   (permanently active)
```

Upon execution of READ NS, the mechanism reads down from the top to first flag that is active [X]. Associated 10 bit address is sent to Nanostore. If the corresponding nanoword is executed (GATE NS), flag is turned off.

Note: The flag associated with NANOBRANCH is treated in a special manner to be described in section 4.5.2.3.

4.5.2.2 PROGRAM CHECK

When a Program Check occurs, the following is automatically done:

1. Execution of the active nanoword is terminated.
2. The appropriate bit is set in the Program Check Status fields to identify the type of error.
3. The contents of RONS[0] are loaded into the Control Matrix to begin execution of the service program.

Since the entry point of RONS[0] is shared by the Program Check service program and the Machine Start program (see section 4.8), the common program must test for Program Check Status fields of zeros (cleared by the Machine Start pushbutton) to determine the nature of its invocation. A "special condition", set to "TRUE" if any of the Program Check Status bits are on, can be tested for this purpose (see section 5.7.1 and 5.7.2).

General Program Check Types are:

1. MS Parity Error
2. MS Address Error
3. Illegal Micro Operation Entry
4. Priviledged Operation(Supervisory) Error
5. Nanoprogram(Microinstruction) Time Out

4.5.2.3 NANOBRANCH

The nanobranch facility is one means of continuing a nanoprogram beyond one nanoword. Due to the high priority given to the nanobranch operation, a branch-connected nanoprogram is never interruptible except by Program Check.

The source of the nanobranch address is a 10-bit field, KN, within the active K-vector.

Control of the nanobranch activity status for priority selection is accomplished through the BRANCH bit in the active K-vector, in conjunction with the READ NS nanoprimitive:

> Each time a READ NS is executed, the BRANCH bit is tested. If active, the nanobranch address is taken. If inactive, the nanostore address is taken from one of the lower priority sources as described in Section 4.5.2.1. Thus, the BRANCH bit serves as the activation flag for the selection of the nanobranch address.

The initial condition of the BRANCH bit is determined by its setting in the nanoword gated into the control matrix. If set, BRANCH is ACTIVE as soon as the nanoword (i.e., the one containing the BRANCH bit) is loaded into the Control Matrix. The state of another bit, the "ALTERNATE" bit in the active K vector determines the future condition of the BRANCH flag. When ALTERNATE is not set, the BRANCH bit retains its initial status. However, when the ALTERNATE bit is set, every execution of READ NS in the active nanoword acquires the secondary function of inverting (complementing) the BRANCH activity flag after initiating the Nanostore read operation.

Thus, the nanoprogrammer can specify four possible settings of these two bits to control the selection of the nanobranch address:

ALTERNATE	BRANCH	ACTION
0	0	NANOBRANCH ADDRESS NEVER USED BY READ NS
0	1	NANOBRANCH ADDRESS ALWAYS USED BY READ NS
1	0	2ND, 4TH, 6TH, ETC. READ NS USES NANOBRANCH ADDRESS
1	1	1ST, 3RD, 5TH, ETC. READ NS USES NANOBRANCH ADDRESS

Note that this control is always determined on the READ NS execution and not on the GATE NS as for other activation flags. Thus the SKIP and/or GATE NS facilities may be used to effect a variety of conditional nanobranching.

4.5.2.4 EXTERNAL INTERRUPTS

Interrupts are signals which can notify the program of the occurrence of events external to the QM-1. A maximum of 30 such signals are directly detectable by QM-1 hardware.

The 30 interrupts are ordered by priority level for Nanostore address selection and are labeled Level 2 through Level 31; Level 2 is highest-priority (immediately below nanobranch), and Level 31 is lowest priority (immediately above the NanoProgram Counter).

Assignment of levels to signal lines is performed by NANODATA at installation time according to user specifications. A typical assignment is shown in the adjoining figure.

For an interrupt to become ACTIVE for priority selection, it must be "LATCHED", "ENABLED", "PENDING" and "ALLOWED".

An interrupt level is LATCHED when a 50 ns. pulse is sensed on its signal line.

Interrupt levels are individually ENABLED by the "1" state of the corresponding Interrupt Enable bit. These 30 bits are stored in External Store registers 18 and 19 (see section 4.2.5).

INTERRUPT LEVEL ASSIGNMENTS

Channel	Level Assigned		
Number	Data In	Data Out	Status
0	2	3	22
1	4	5	23
2	6	7	24
3	8	9	25
4	10	12	26
5	13	11	27
6	14	15	28
7	16	17	29

Levels 18-21, 30 and 31 may be assigned to other external signals.
Levels 2-11 - Nano Interrupts.
Levels 12-31 - Micro Interrupts.

Every ENABLED level is tested for the presence of a LATCHED interrupt signal by each execution of GATE NS. If this test succeeds, the level is set to PENDING status. Once a level is PENDING, it remains in that state until the priority-select mechanism eventually selects the Nanostore address corresponding to that interrupt level, and the associated nanoword is loaded into the Control Matrix to begin the service program; at that time the level is also unLATCHED, and unPENDING.

A PENDING interrupt level automatically becomes ACTIVE for priority selection if and only if its associated ALLOW INTERRUPT bit in the active K-vector is "1" when the priority-select mechanism is invoked by READ NS. There are two such bits ("ALLOW NANO INTERRUPT" and "ALLOW MICRO INTERRUPT").

The high 10 interrupt levels (2-12) are designated as NANO INTERRUPT levels. The remaining 20 levels (12-31) are designated as MICRO INTERRUPT levels.

If a nanobranch is not taken at the end of executing a nanoword, and if no interrupts are active, the priority-select mechanism gives control to the NanoProgram Counter (see Figure 4.5.2.1A); hence the ALLOW INTERRUPT bits are a facility the programmer can use to insure that a chain of nanoword executions invoked through the NPC is not interrupted. This subject is further discussed in section 4.5.4.2.

All I/O interrupts may be blocked from priority selection by disabling I/O interrupts with the Auxilliary Action "disable" command as described in Section 5.8.2. The ALLOW INTERRUPT bits are then ignored and no I/O interrupts will be accepted. Following the Auxillary Action "enable" command, all blocked I/O interrupts again become eligible for priority selection, assuming all other prerequisites exist, as described above.

The Nanostore addresses associated with the interrupt levels are generated from six-bit fields in ten External Store registers 22 - 31 (see section 4.2.5). The mapping of the six-bit field into the 10-bit address is as shown on the right. This mapping permits up to 16 interrupt entries in each page of nanostore. To conserve entry points in Nanostore, several interrupt levels may be assigned the same address in nanostore by placing the common address in the appropriate positions in External Store registers 22 - 31.

MAPPING OF INTERRUPT ADDRESSES

SIX-BIT FIELD:	"ABCDEF"
TEN-BIT ADDRESS:	"0AB000CDEF"

Finally, a facility exists for the programmer to "generate" interrupts (simulate external interrupts) and also to "clear" interrupt latches. One interrupt level can be so affected in a nanoword. When the "GENERATE INT" bit is set in the active K-vector, the selected level is latched or unlatched at the beginning of T-period 1: if the "generate" option is used, the level becomes LATCHED, PENDING, and ACTIVE in time to be selected for execution by a READ NS executed in T-period 3 or later; if the "clear" option is used, the level is unLATCHED immediately upon activation of the current nanoword. The mechanism for selecting the level and the "generate" vs. "clear" option is presented in section 5.8.1.

Other External Interface facilities are discussed in section 4.6.

4.5.3 NANOPROGRAM COUNTER

4.5.3.1 GENERAL

When an nanobranch is not taken and no interrupts are active, the priority-select mechanism supplies an address to Nanostore from the NanoProgram Counter, the lowest element on the priority list.

The NPC is a 10-bit register which changes value only as a result of nanoprimitive commands. The following (mutually exclusive) NPC control operations are available in the T-vector (all trailing-edge):

LOAD NPC (CS)
LOAD NPC (KN)
LOAD NPC (SEQUENCE)

The first operation -- LOAD NPC (CS) -- involves microinstruction execution, since the address is a Control Store Opcode. This is discussed in the next Section.

The LOAD NPC (KN) operation loads the NanoProgram Counter from the KN field in the active K-vector. Thus an executing nanoword can transfer nanoprogram execution to NS(KN) either directly (nanobranch) or through NPC (NPC branch); the interrelationship of these two facilities is discussed in section 4.5.4.

The LOAD NPC (SEQUENCE) operation adds one (modulo 1024) to the contents of the NPC. Thus a nanoprogram executing at an NPC-specified Nanostore address can conveniently continue execution through sequential Nanostore locations (nanosequencing).

4.5.3.2 MICROINSTRUCTION EXECUTION

One of the most important modes of program control is the invocation of a nanoprogram by a microinstruction; the operation code of a machine micro-instruction, extracted from Control Store, is used to select the Nanostore entry address of the nanoprogram (of one or more nanowords) whose execution defines that microinstruction.

When the nanoprimitive command LOAD NPC (CS) is executed, the following occurs (trailing-edge):

a) The high-order three bits of NPC are loaded with the Nanostore Page Index from the low-order three bits of FIDX, a special F-register (see section 4.3.2.3);
b) the low-order seven bits of NPC are loaded from the high-order seven bits of the COD bus;
c) the low-order eleven bits of the COD bus are saved in a dedicated register.

The nanoprimitive "LOAD R31" is available to cause the following action:

a) The high-order seven bits of R31 are cleared to zeros; and
b) the low-order eleven bits of R31 are loaded with the saved eleven low-order bits of COD (this is the parameter part of the machine microinstruction).

This event is concurrent with six-bit transfers executed in the last T-step of the previous active nanoword; thus the new contents of the C, A, and B fields in R31 are available for gating to F-store in the first T-step of the microinstruction, if desired (see section 5.3.5).

The high-order seven bits of a machine microinstruction are thus defined as the micro-opcode, and provide the microprogrammer with a maximum of 128 microinstructions supported by a page of Nanostore; i.e., 128 NPC-addresses are possible under a given value of the Nanostore page index in FIDX. Different Nanostore pages may be used to define different micro-machines, extend the microinstruction set of a given micro-machine, implement different machine states, and/or contain continuations of nanoprograms from another page. For convenience in microinstruction sequencing, one of the Local Store MPC's will normally be used in addressing Control Store to generate the machine microinstruction on the COD bus (see section 4.2.4).

To protect against the execution of illegal micro-opcodes (for example, when some or all nanowords in a page are used for nanoprogram continuations and/or interrupt entry points), a "LEGAL MICRO ENTRY" bit is provided in the K-vector. If this bit is off ("0") in a nanoword loaded into the Control Matrix as the initial word of a nanoprogram invoked by a microinstruction, a Program Check is generated.

To protect against infinite looping between two or more nanowords, a Microinstruction Time-Out facility generates a Program Check if microinstruction executions do not follow within approximately one second of each other.

This section has used the term "machine microinstruction" to refer to a Control Store word which is executed through NPC to invoke a nanoprogram. For the microprogrammer, however, a "microinstruction" may consist of several Control Store words and contain a large number of parameters and/or immediate operands; the only restriction is that one of the words (most conveniently, the first of a contiguous string) must be a machine micro-instruction. The invoked nanoprogram is able to fetch the other words from Control Store, and, if appropriate, can use the C,A, and B fields of the R31 interface to route six-bit parameters to various control registers.

Microinstructions defined using this technique can be quite powerful, and have the advantage of economizing on micro-opcodes; for example, a "general arithmetic and logic" microinstruction can be defined by routing a parameter field to the ALU control register (KALC, in the active K-vector).

4.5.4 NANOPROGRAM FLOW

4.5.4.1 NANOPROGRAM CONNECTION

Nanoprogram execution controlled by the NPC may be considered to be "nanoprogram mainline" flow. A mainline is ordinarily initiated by the invocation of a machine microinstruction (section 4.5.3.2).

Using this model, the operation LOAD NPC (SEQUENCE) then can be used as a straightforward method of continuing a mainline; the operation LOAD NPC (KN) has the effect of transferring the location of the mainline to a different place in Nanostore. Thus the following technique can be used to maximize the number of micro-opcodes in a page: If a nanoprogram which defines a microinstruction is longer than one nanoword, the first nanoword exits by transferring the mainline to a different page of Nanostore; the nanoprogram consumes only one micro-opcode entry point in the initial page.

Since the nanobranch facility does not affect the state of the NanoProgram Counter, the following technique provides a mainline nanoprogram with the capability for calling one level of subroutines in Nanostore, as follows: A call is effected by a nanobranch to the first word of the sub-nanoprogram, which must proceed (if longer than one word) by nanobranch only; the called sub-nanoprogram returns to mainline via the NPC, which has remained as a link.

A sub-nanoprogram can terminate the nanoprogram -- even conditionally, if desired, since its return to nanoprogram mainline is exactly the same as those steps of a return to microinstruction control that follow LOAD NPC (CS). For example, the mainline can perform normal microinstruction prefetch operations and then conditionally (via SKIP) execute LOAD NPC (CS) before calling the sub-nanoprogram; the latter will return either to mainline or to new microinstruction control, depending upon whether LOAD NPC (CS) was SKIPped or not, respectively.

4.5.4.2 INTERRUPTIBILITY

One suggested mode of interruptibility is to allow low-priority interrupt levels to take control only between microinstructions; this plan is enforced by setting the ALLOW MICRO INTERRUPT bit (in the K-vector) only in the exiting nanoword of a nanoprogram (i.e., that word which is ordinarily succeeded by a nanoprogram invoked by the next machine microinstruction). In this mode, the interrupt-service nanoprograms are free to make use of the NPC to establish a mainline, and thus call subroutines, transfer to microinstruction control, etc.; the lower-priority interrupts are more likely to require such service (e.g., end of IO operation).

A suggested parallel mode of interruptibility is to allow mainline nanoprograms to be interrupted between (some) nanowords, using the ALLOW NANO INTERRUPT bit. (Note: branch-connected nanoprograms, including sub-nanoprograms as defined in section 4.5.4.1, cannot be interrupted because of the high fetch-priority of nanobranch.) When allowing the mainline nanoprogram to be interrupted, the interrupt-service nanowords must proceed by nanobranch only, since any other technique would destroy the value of the NPC and hence break the link for returning to the interrupted (mainline) nanoprogram. The higher-priority interrupts are more likely to be serviceable by this kind of program (e.g., single-word transfer in a data stream).

Any programming structure must allocate various machine resources to the various levels of program control. For example, programming conventions could be established such that bus controls are undefined between microinstructions (for free usability by interrupt service routines), but are expected to hold between mainline nanowords; that certain G's do not change value between microinstructions; etc.

4.5.4.3 HOLD

For convenience in programming across nanowords, the "HOLD" and "HOLD 2" bits in the active K-Vector are provided. They allow the nanoprogrammer to retain various control values in the K vector portion of the Control Matrix during the transition to the next nanoword.

If a HOLD bit is set ("1") in the active nanoword, then the corresponding K fields in the control matrix do not change their values as a result of gating the next nanoword into the control matrix. The action of the HOLD bits is suppressed if the next nanoword is invoked by microinstruction entry (GATE NS and LOAD R31 in the final T-Vector) or by program check interrupt.

For obvious reasons, the HOLD control and the ALLOW INTERRUPT controls should normally be used with mutual exclusion.

The following is a list of K-Vector fields affected by the HOLD bits:

HOLD	HOLD 2
KALC	KA
KSHC	KB
KSHA	
KS	

4.6 EXTERNAL INTERFACE

The material in this section functionally specifies the interface between the QM-1 computer itself and its environment. Information on NANODATA Channel Control Units is presented in a Section 8 of this manual.

The external interface consists of eight "external ports", each identified by its association with a Port Register (E0 through E7; see section 4.2.5).

The following "outgoing" external interface facilities are bused for common use by the eight ports:

a) The "Phantom Bus" (current input to the "phantom" register, FIPH) supplies six bits of information

b) The "G-bus" supplies six bits of information taken from one of 16 sources: G0 through G11, KSHA, B, KS, KX. The selection of the source is performed by the "GSPEC" field in the currently active T-vector, and is further discussed in section 5.5.2 (where the value on the G-bus is referred to as "G(GSPEC)").

c) IO Clock - a syncronizing signal to external devices available at the port during each T-step.

d) XIO Strobe - a syncronizing signal to external devices, generated only when XIO is present.

e) MASTER CLEAR - a signal sent when the system is initially cleared; this signal cannot be generated by program control.

The following "outgoing" external interface facilities are local to each port:

a) A path through which an external unit can read the contents of the Port Register (18 bits in parallel).

b) The "Port-XIO" pulse.

c) The "Port-RIO" pulse.

The following "incoming" external interface facilities are also local to each port:

a) A path through which an external unit can supply data to the Port Register (18 bits in parallel).

b) The "IO ID" lines, through which an external unit can supply six bits of information to the port.

c) Some number of interrupt levels, logically assigned to the port by software in accordance with the physical system configuration.

Program control of the external interface involves these internal facilities:

a) KA (of the active K-Vector),
b) The RIO nanoprimitive ("Read IO"),
c) The XIO nanoprimitive ("transmit IO"),
d) Six-bit transfer nanoprimitives, used to read IO ID,
e) The interrupt structure, as presented in section 4.5.2.4.

KA is used modulo 8 to select one of the eight ports for nanoprimitive control. If no external unit interfaces to the KA-selected port, incoming values are zero and outgoing operations are null.

Execution of the XIO nanoprimitive causes a Port-XIO signal to be sent through the KA-selected port for the duration of the XIO, and XIO Strobe to be sent to all ports.

Execution of the RIO nanoprimitive has these effects:

a) At leading edge, a Port-RIO pulse is sent through the KA-selected port.

b) At leading edge, the KA-selected Port Register is set to zeros.

c) At trailing edge, the contents of the 18 incoming data lines associated with the latter register are gated into that register.

The 6-bit IO ID of the currently interrupting device is available to the program as an AUX (see section 5.5.2, and section 8).

4.7 WRITING NANOSTORE

Each 360-bit word in Nanostore is partitioned into 20 18-bit bytes for the purpose of writing. When the WRITE NS nanoprimitive is to be used,

the Nanostore address is taken from the 10 bits of R31 on the high-order side of B;

the byte selection is the B field, used modulo 32;

the 18 bits of data to be written are taken from the EOD bus.

Each time an address is sent to Nanostore (with a Read NS or a WRITE NS), 160 ns later the full 360 bits is available for GATE NS. Therefore, if WRITE NS is followed by GATE NS, the full Nanoword with the modification will be gated into the Control Matrix, and execution begun in T1.

If either a nonexistent word-location is addressed, or B is greater than 19 (bytes are addressed 0 through 19), WRITE NS does not alter Nanostore. Instead it acts like READ NS; it calls out zeroes on a bad word address, and a non altered word on a bad byte address.

If WRITE NS and READ NS appear in the same active T-vector, READ NS is ignored.

4.8 READ-ONLY MEMORIES AND MACHINE START

In addition to Nanostore and Control Store, the QM-1 contains a Read-Only Nanostore (RONS) of 32 360-bit words, and a Read-Only Control Store (ROCS) of 128 18-bit words. These memories are logically distinct from NS and CS, and are accessed as follows:

When the Nanostore Mode switch in FIDX is cleared ("0"), RONS is inaccessable. When set ("1"), RONS address spaces are effectively subtituted for NS address spaces on READ NS.

When bit 17 of a Control Store address is cleared ("0"), ROCS is inaccessible. When set ("1"), ROCS addresses spaces are substituted for Control Store address spaces on READ CS or WRITE CS; the WRITE will not alter CS but will instead act like a READ CS. NOTE: If incrementing a Control Store address causes a "negative" result, ROCS will be accessed. A READ CS from a nonexistent ROCS address places ONES onto the COD bus.

The contents of RONS AND ROCS are specified by the user and permanently inserted by NANODATA at installation time. (NANODATA-supplied machine diagnostic routines must be included, and NANODATA-supplied system software may be specified.)

Machine use of these memories is for nanoprogram entry at RONS[0] for Program Check (section 4.5.2.2) and for Machine Start.

When the QM-1 MASTER CLEAR / START button is depressed,

a) FIDX is cleared;
b) the Program Check Status fields are cleared so that the program starting at RONS[0] can recognize its invocation by Machine Start rather than Program Check (see section 4.5.2.2); and
c) RONS[0] is fetched and loaded into the Control Matrix to begin execution.

5 QM-1 FUNCTIONAL SPECIFICATIONS, PART II

5.1 GENERAL

Sections 4 and 5 of this manual are a complete functional specification of the QM-1 CPU in two parts. Part I (Section 4) has explained QM-1 concepts, architecture and operations. It has provided an overview of all of the features of the machine. Part II (Section 5) is intended as a programmer's reference guide and will complete the description of those parts of the machine covered only briefly in Part I.

It is assumed that the reader has a general understanding of the QM-1 at this point. Thus Part II will concentrate more on the detailed operation of the individual machine functions and less on their possible combined use.

The next two sections present, in summary form, all of the control functions included in the machine. In most cases, the functions are activated by single bits or contain a string of bits used as a numeric value. In the few remaining cases, the encodings of the bits are given. References are provided for each function to the section numbers where the function is specified in detail.

5.2 SUMMARY OF NANOPRIMITIVE CONTROLS

5.2.1 K-VECTOR CONTROL FIELDS

The control function of each of the fields in the K-vector is summarized in the table below, along with references to sections in which the function is described. (The number of bits in the field is shown in parentheses.)

CONTROL FIELD	(Bits)	SUMMARY OF CONTROL FUNCTION	References
KN	(10)	Address of possible successor nanoword. Nanobranch address and source for NPC load.	4.5.2.3
SUPERVISOR	(1)	Program Check if on when this word is invoked while not in Supervisor Mode.	4.5.2.2
LEGAL MICRO ENTRY	(1)	Program Check if not on when this word is invoked by a microinstruction.	4.5.3
BRANCH	(1)	Must be on if nanobranch planned from this word. Complemented after each READ NS when ALTERNATE is on.	4.5.2.3
ALTERNATE	(1)	Causes BRANCH to be complemented after each READ NS.	4.5.2.3
HOLD	(1)	Inhibits automatic loading of KALC, KSHC, KSHA, and KS from next nanoword to be executed, unless executed by microinstruction or Program Check.	4.5.4.3
HOLD 2	(1)	Inhibits automatic loading of KA and KB from next nanoword to be executed, unless executed by microinstruction or Program Check.	4.5.4.3
ALLOW NANO INTERRUPT	(1)	Allows higher-priority interrupts at end of execution of this word, if nanobranch is not taken.	4.5.2.4 4.5.4.2

ALLOW MICRO INTERRUPT	(1)	Allows lower-priority interrupts at end of execution of this word, if nanobranch is not taken.	4.5.2.4 4.5.4.2
GENERATE INTERRUPT	(1)	Generates or clears an interrupt level according to G[GSPEC] in T1.	4.5.2.4 5.8.1
ALU STATUS ENABLE	(1)	Enables move of C,S,R,O bits from local to global upon GATE ALU; C treated specially.	5.6.2
SH STATUS ENABLE	(1)	Enables move of SHB, SLB bits from local to global upon GATE SH.	5.6.3
DIRECT MS ACCESS	(1)	Inhibits MS base addressing and field length protection in this nanoword.	4.2.6.3
KA	(6)	Constant and/or scratch field for nanoword; source and destination AUX.	4.3.3
KB	(6)	Constant and/or scratch field for nanoword; source and destination AUX.	4.3.3
KALC	(6)	ALU control; destination AUX.	5.6.2
KSHC	(6)	Shift control; destination AUX.	5.6.3
KSHA	(6)	Shift amount; destination AUX.	5.6.3
KS	(6)	Global condition (and general) test mask; source and destination AUX.	4.6 5.7.1
KT	(6)	Local condition test mask (also constant and/or scratch); source and destination AUX.	5.7.1
KX	(6)	Special condition test mask (also constant and/or scratch); source and destination Aux.	5.7.1
SPARE	(2)	Reserved for future use	
	----		
	72 BITS		

5.2.2 T-VECTOR CONTROL FIELDS

The control function of each of the fields in the active T-Vector is summarized in the table below, along with references to sections in which the function is described. A code showing the characteristic timing of the action associated with the function is given; LE = Leading Edge, TE = Trailing Edge. (The number of bits in the field is shown parenthetically.)

CONTROL FIELD	BITS	SUMMARY OF CONTROL FUNCTION	TIME	Refs.
STRETCH	(1)	Stretches time of this T-step from one T-period to two.		4.4
WRITE NS	(1)	Writes 18 bits from EOD bus into Nanostore	LE	4.7 5.4.1.2
XIO	(1)	Sends pulse to external interface; one of eight external ports selected by KA.	LE	4.6
RIO	(1)	Clears Port Register and sends pulse through port, then gates external data word into Port Register; selected by KA.	LE	4.6
MSGO	(1)	Initiates MS operation; split-cycle if alone, full-read if MSRS simultaneous.	LE	4.2.6.2 5.4.3
MSRS	(1)	If alone, requests second half-cycle of MS split-cycle operation; if with MSGO, initiates full-read.	LE	4.2.6.2 5.4.3
GATE MS	(1)	Gates MOD bus into Local Store or Port Registers; modified by RMI SELECT.	TE	4.2.6.1 5.4.3
RMI SELECT 00 BYPASS 01 PARAMETER SET A 10 PARAMETER SET B 11 PARAMETER SET C	(2)	Selects RMI parameters for GATE MS, including BYPASS. If RMI not installed all encodings are BYPASS	LE	4.2.6.4
GATE ES	(1)	Gates EOD bus into Local Store.	TE	4.2.5

LOAD ES	(1)	Loads an External Store register from EID bus.	TE	4.2.5
TXX	(1)	Halts T-Clock with Program Step Switch.	TE	5.8.3
READ CS	(1)	Reads Control Store; uses CS ADDR SELECT.	LE	4.2.4 5.4.2.2
WRITE CS	(1)	Writes Control Store; uses CS ADDR SELECT.	LE	4.2.4 5.4.2.3
CS ADDR SELECT 000 CIA 001 COD 010 MPC 011 MPC+1 100 MPC+2 101 MPC+B 110 MPC+AB 111 INDEX	(3)	Selects address for READ CS, WRITE CS. (MPC is selected by FMPC) A and AB are sign extended operands. INDEX is output of INDEX ALU.	LE	4.2.4 5.4.2.1
GATE CS	(1)	Gates COD bus into Local Store.	TE	4.2.4 5.4.2
GATE ALU	(1)	Gates ADD bus into Local Store.	TE	4.2.3
GATE SH	(1)	Gates SOD bus into Local Store.	TE	4.2.3
CARRY CTL 000 NO OPERATION 001 CLEAR CIH 010 SET CIH 011 ALU TO BOTH 100 ALU TO COH 101 SET COH 110 CLEAR COH 111 SH TO COH	(3)	Controls Carry operation within the ALU and Shifter components.	TE	4.2.3.4
INDEX	(1)	Gates INDEX ALU output into Local Store, selected by G(GSPEC).	TE	4.2.2.3 5.6.4

INC MPC	(1)	Increments MPC selected by FMPC; modified by GSPEC.	TE	4.2.2 5.4.2.1 5.6.5
LOAD NPC 00 NO OPERATION 01 (CS) 10 (KN) 11 (SEQUENCE)	(2)	Loads or sequences NanoProgram Counter.	TE	4.5.3 4.5.4
READ NS	(1)	Reads NS; address is from priority-select mechanism. Influences BRANCH.	LE	4.5 5.4.1.1
GATE NS UNCON- DITIONALLY	(1)	Causes the nanoword last read to be gated into the Control Matrix. Independent of any TEST ACTION in T.	TE	4.5.1 5.5.1
TEST ACTION 0 SKIP 1 GATE NS	(1)	Conditional Action based on Test Specifier	TE	4.5 4.5.1 5.7.2
TEST SPECIFIER 000 NEVER 001 ALWAYS 010 If FIST AND KS = 0 011 If FIST AND KS NOT = 0 100 If LOCAL CONDS AND KT = 0 101 If LOCAL CONDS AND KT NOT = 0 110 If SPECIAL CONDS AND KX = 0 111 If SPECIAL CONDS AND KX NOT = 0	(3)	Specifies the conditions under which TEST ACTION is to be executed	LE	5.7.1
LOAD R31	(1)	Enables R31 to be loaded with micro-instruction parameters.	TE	4.5.3.2 5.3.4
AUXILLARY ACTION	(1)	Initiates Action specified by the contents of FACT (F register 14).	LE	4.3.2.3 5.8.2

```
GSPEC                  (4)  Selects a G or pseudo-G for 6-bit        5.5.2
   0000  G0                 transfers, right input to ALUF,
   ----  ---                used in GENERATE INTERRUPT, External
   1011  G11                Interface G-lines; also used with
   1100  KSHA              INC MPC.
   1101  B
   1110  KS
   1111  KT

FSEL0                  (5)  Selects F register for 6-bit transfers   5.5.2
FSEL1                  (5)  in Group 0, 1, and 2 respectively.
FSEL2                  (5)

AUX0                   (3)  Selects AUX for 6-bit transfers in
AUX1                   (3)  Group 0, 1, and 2 respectively.
AUX2                   (3)  (AUX2 applies to Group 2 input,
AUX3                   (3)    AUX3 applies to Group 2 output.)

IN0                    (1)  Commands AUX into F register transfer
IN1                    (1)  using AUX0, AUX1, AUX2 to FSEL0,
IN2                    (1)  FSEL1, FSEL2 respectively.

OUT1                   (1)  Commands F register output to AUX
OUT2                   (1)  transfer using FSEL0, FSEL1, FSEL2 to
OUT3                   (1)  AUX0, AUX1, AUX3 respectively.

                       ---
                       72 Bits
```

5.3 FUNDAMENTAL TIMING CONSIDERATIONS

5.3.1 GENERAL

The "hardware level" QM-1 is a highly parallel machine. One of the tasks facing the nanoprogrammer is to put together the functions he desires in such a way as to utilize this parallelism to the fullest extent possible. Hence he must have an intimate knowledge of the internal timing of the machine. This section on timing considerations is included in order that nanoprogrammers can answer questions regarding meaningful combinations of functions in the same or adjacent T-steps.

All T-vector control functions have been classified as "Leading Edge" (LE) or "Trailing Edge" (TE) functions depending on the time of the action they initiate, relative to the period of the T-step in which the control is active. The period of any T-step is defined as the time between the machine clock pulse which causes the T-vector to become active and the next clock pulse which causes the next T-vector to become active. These pulses are known as T-clock pulses or just T-Clocks and the T-step (T2 for example) is as shown.

```
T-CLOCK  I        I        I
 PULSES  I        I        I
  ------ -------- -------- --------

                  --------
T-VECTOR   NOT    I   T2   I  NOT
  T2      ACTIVE  I ACTIVE I ACTIVE
         --------          --------

                  T-STEP T2
```

Leading edge functions are those which are triggered by the beginning edge (or activation) of the T-step and trailing edge functions are those triggered by the ending edge (or deactivation) of the T-step.

Examination of this situation for two successive T-steps shows that a trailing edge for one T-step occurs at exactly the same time as the leading edge of the next. Thus it would seem that a leading edge event could occur at exactly the same time a trailing edge transition is happening. In actuallity, this problem is avoided by having some functions "more trailing edge" than others. This is necessary since the Machine State Vector (active K and T-vectors) must be in a defined state before the 6-bit domain can operate properly. And the 6-bit sections must be in a defined state for the 18 bit domain to operate properly. Hence the necessity of two additional clocks derived from the T-Clock. These are, not suprisingly, called the "F Register Clock" (or F-Clock) and the "LS Register Clock" (or R-Clock). The actual delays between these clocks are important only when the boundaries between the three domains within the machine are crossed. The extreme case is in R31 operations since all three domains meet in R31 (covered extensively in Section 5.3.5).

Briefly, the clocks act as follows. The T-Clocks activate a T-Vector. All leading edge functions are begun immediately. All decoding and set-up for trailing edge functions also begins at this time. The next T-Clock deactivates this T-Vector (which will now be called the previous T-Vector) and activates the next one. Approximately 20 nanoseconds after the T-Clock, the F-Clock occurs, completing any 6-bit data transfers specified in the previous T-Vector. Approximately 20 nanoseconds after the F-Clock, the R-Clock occurs. This completes any 18-bit transfers specified in the previous T-Vector. This sequence is shown in Figure 5.3.1A.

```
                         ----------
        ACTIVE          I          I
       T-VECTOR         I          I
     -------------------            ----------------------

                         <--Start Leading Edge Functions
                        I          I
       T-CLOCK          I          I
     ------------------- ---------- ----------------------

                                      <--Finish F Transfers
                            I          I
       F-CLOCK              I          I
     ----------------------- ---------- ------------------

                                          <--Finish R Transfers
                                I          I
       R-CLOCK                  I          I
     --------------------------- ---------- --------------
```

RELATIONSHIP OF MACHINE CLOCKS   Figure 5.3.1A

5.3.2 LEADING EDGE FUNCTIONS

The Memory Reference Functions; Read/Write Nanostore, Read/Write Control Store, Go/Restart Main Store, are all operations which do not cause data to be gated but are necessary to make data available for gating. In order to make the data available as soon as possible, these operations must be initiated as soon as the T-Vector in which they are specified becomes active. Thus they are leading edge functions. Care must be taken to assure that the address and/or data to be used by the operation is stable before the function is initiated. Section 5.4 covers this in detail.

XIO and RIO are the only other leading edge functions. Both XIO and RIO generate a signal to the External Port and must therefore begin on the leading edge in order for the action they initiate to be completed by the end of the T-period.

5.3.3 TRAILING EDGE EVENTS

5.3.3.1 T-CLOCK EVENTS

LOAD NPC is executed on the trailing edge T-clock. The three possible sources of new values to be transferred into the NanoProgram Counter are the COD bus, KN in the executing nanoword, and the current value of NPC. No special timing problems arise with the LOAD NPC operation.

GATE NS and LOAD R31 are the only other functions executed on the trailing edge T-Clock. Both are used to cause the transition between one instruction and the next. Thus they must be completed before any other functions can begin.

A conflict is possible between each of these functions and some other function in the machine. In the case of GATE NS, an F-transfer into a K, initiated in the same T-step as the GATE NS will cause an undefined result in the K unless the K is "held" by the appropriate HOLD command. See section 4.5.4.3.

In the case of LOAD R31, an F transfer into R31, initiated in the same T-step as LOAD R31 will cause an undefined result in R31. See Section 5.3.4. An R transfer will override the effect of the LOAD R31.

5.3.3.2 F-CLOCK EVENTS

All F transfers are completed by the trailing edge F-Clock. These include F Register Increment and Decrement and ALUF operations since the results are gated as an F transfer.

Simultaneous F transfers to the same F-Register do not cause undefined results since they occur at exactly the same time. A logical "OR" of the transferred values occurs.

5.3.3.3 R-CLOCK EVENTS

All 18 bit transfers into Local Store or External Store are syncronized on the trailing edge R-Clock. Simultaneous R transfers into the same Local Store register will produce the logical "OR" of the transferred values.

5.3.4 R31 OPERATIONS

R31 is the primary interface between the 18-bit architecture and the 6-bit architecture in the QM-1, since it is simultaneously a Local Store register and also contains three 6-bit AUX fields:

```
      ------------------------------------------------------
R31   I            18 Bit Local Store Register             I
      ------------------------------------------------------


      ------------------------------------------------------
R31   I        C        I        A        I        B       I
      ------------------------------------------------------
 Bit   17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
```

When serving in its additional special function as the Micro Instruction Register, R31 is classified in the control matrix domain, since the LOAD R31 command transfers the saved microinstruction parameter part into R31 on the Leading Edge T-Clock:

```
      ------------------------------------------------------
R31   I     7 ZERO BITS    I     A        I        B       I
      ------------------------------------------------------
```

Because of this interface, care must be taken in organizing transfers involving R31. The following programming rules are derived from the clock relationships discussed in Section 5.3.1.

1. If the only transfers into R31 commanded in a T-step are in the 18 bit domain, then six-bit transfers out of R31 commanded in the following T-step will occur too soon to use the 18-bit value, and will instead transfer an undefined value. This holds true unless the T-step containing the six bit transfer command is stretched, in which case such transfers do use the new 18-bit value.

2. If the only transfers into R31 commanded in a T-step are from the six-bit domain, these values are available for transferring out to either domain in the next T-step.

3. If transfers into R31 from both the six-bit and the 18-bit domains are commanded in a T-step, an undefined value results in R31.

4. When R31 is used within one domain only, normal timing rules apply, as in section 4.4.

5. If six-bit transfers into R31 are concurrent with LOAD R31 parameter loading (i.e., if such transfers are commanded in the last T-step of a nanoword along with the LOAD R31 command) an undefined value results in R31.

5.4 MEMORY REFERENCE FUNCTIONS

5.4.1 NANOSTORE OPERATIONS

5.4.1.1 READ NS

READ NS is a leading edge command that causes a 360-bit word to be read from nanostore using the nanostore address selected previously by the nanostore addressing mechanism (section 4.5.2). The address must have been established in the previous T-step. This address must be stable for the T-step in which READ NS occurs. Neither of these requirements cause any difficulty since the LOAD NPC commands are properly syncronized to satisfy them.

For Nanostore data-out to be available in time for a trailing-edge GATE NS executed in a given T-period, the leading-edge operation READ NS must occur in the prior T-period or earlier. Therefore READ NS, GATE NS can be programmed as a sequence in one T-step if and only if that T-step is STRETCHed.

READ NS, GATE NS executed in the same un-STRETCHed T-step result in an undefined value loaded into the Control Matrix.

READ NS cannot be commanded in T1 of any nanoword that ALLOWs INTERRUPTs. Undefined data results in this situation.

5.4.1.2 WRITE NS

WRITE NS is a leading-edge nanoprimitive that initiates writing 18 bits into nanostore from the EDD bus at the address specified by the contents of R31. The B field of R31 specifies which of the 20 bytes of the nanoword is to be written (0-19, modulo 32). The 10 bits of R31 on the high order side of the B field in R31 select the paricular nanoword to be written. With an invalid nanostore address (out of range), WRITE NS does not alter Nanostore.

```
     ------------------------------------------------------------
R31  IXX XX <-----NANOWORD ADDRESS------> XX  BYTE ADDRESS I   Bits marked XX
     ------------------------------------------------------------  are ignored.
      17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
```

The WRITE NS address must be stable for 3 T-periods. (Attempts to initiate a READ NS or another WRITE NS during such time are ignored.) Thus if WRITE NS appears in T-period T(n), Nanostore will be written in time for a READ NS executed in T-period T(n+2). The WRITE NS acts like a READ NS and brings up the full modified Nanoword ready to be gated.

Assuming that WRITE NS is executed in T-period T(n), results of the operation are undefined if:

1. The Nanostore word-address is modified by a command in T-period T(n-1).
2. The byte-selector in B is modified by a command in T-period T(n-1).
3. The data on the EDD bus is modified by a command in T-period T(n-1).

It is possible to execute a nanoprogram by addressing it from R31 and using the "WRITE NS" primitive. This is accomplished by putting an invalid byte address in bits 0-15 of R31 and executing a "WRITE NS". The addressed nanoword is not changed but the outputs are available for "GATE NS" into the control matrix.

5.4.2 CONTROL STORE OPERATIONS

5.4.2.1 CONTROL STORE ADDRESS SELECTION

For either READ CS or WRITE CS, the address used is determined by the value of the CS ADDR SELECT field in the same T-Vector as the READ CS or WRITE CS. Since both READ CS and WRITE CS are leading edge functions, the address must be stable at the beginning of the T-step in which the command occurs. This has different implications, depending on the address source. Each of the cases is covered below, assuming that the READ CS or WRITE CS occurs in T(n):

CS ADDR SELECT			CONDITIONS
0	000	CIA	Address is taken from the local store register designated by FCIA. No commands changing FCIA or CIA should appear in T(N-1)
1	001	COD	Here the address is taken directly from the COD bus rather than from a register. Since the only thing that can change the data on the COD bus is a previous READ CS, this should not occur in T(N-1) unless STRETCHed.
2	010	MPC	Address is taken from the local store register designated by FMPC. No commands changing FMPC or MPC should appear T(N-1).
3	011	MPC+1	Here the address depends on FMPC, MPC and the output of the MPC Increment facility. Again, nothing that changes either FMPC or MPC should occur in T(N-1).
4	100	MPC+2	
5	101	MPC+B	In this case, an added factor is involved - the contents of R31. Thus nothing that changes the contents of R31 should appear in T(N-1)
6	110	MBP+AB	
7	111	INDEX	Output from the INDEX ALU; inputs must appear in T(n-1), and must remain stable for three T-Periods.

The Control Store Address determined by the above selection must be stable only for the duration of the T-step in which READ CS or WRITE CS occurs. Thus it is possible to specify in the same T-step, any Trailing Edge operations that change the address.

Both READ and WRITE CS place the 18 bit value on COD. Attempts to read or write nonexistant locartions result in zeros on the COD bus, but does not alter CS.

If READ CS and WRITE CS occur simultaneously, only the WRITE CS occurs. READ CS and/or WRITE CS commands are valid in two successive T-steps only if the first T-step is STRETCHed. Negative CS addresses indicate ROCS.

5.4.2.2 READ CS

READ CS reads the 18 bit value from Control Store at the address determined by CS ADDR SELECT. The value read is placed on the COD bus for gating into a Local Store register, for loading as a microinstruction, for use as an indirect Control Store address or for use as an arithmetic operand. If the READ CS is executed in T-period T(n), the desired data is available on COD for each of these uses at the trailing edge of T(n+1). It is available at the trailing edge of T(n) if the T-step is STRETCHed.

Once established, a COD value remains available until changed by the next Control Store access (READ or WRITE).

5.4.2.3 WRITE CS

WRITE CS writes the 18 bit value from the Local Store register specified by FCID into Control Store at the address determined by CS ADDR SELECT.

In addition to all of the rules that apply to the address selection (section 5.4.2.1), the data on the CID bus must be stable for the duration of the T-step in which the WRITE CS occurs. Thus, for a WRITE CS in T(N) to be valid, nothing that changes either FCID or CID may appear in T(N-1). WRITE CS acts as READ CS and places the new value on COD.

5.4.3 MAIN STORE OPERATIONS

5.4.3.1 READ MAIN STORE

A READ MAIN STORE operation is initiated by placing the appropriate values in the following registers and simultaneously issuing the commands MSGO and MSRS.

FMIX	- pointer to Local Store register used for Main Store Address
MIX	- Main Store Address in Local Store Register
E16	- Main Store Base Address in External Store register
E17	- Main Store Field Length in External Store register

These four values must be changed no later than the T-step before the commands MSGO and MSRS appear. The values need only be held stable during the T-step in in which MSGO and MSRS are issued. MSGO and MSRS must not be issued together until a test of "MS BUSY" indicates that Main Store is available.

The word in Main Store, addressed by the sum of the values in MIX and E16 is read and placed on the MOD bus in time for gating with the trailing edge command GATE MS in the same T-step that the test "MS DATA" indicates data available. If the value in MIX exceeds the value in E17, or if the addressed word is beyond the range of installed Main Store addresses, the result on the MOD bus is zero and a Program Check occurs.

If desired, the participation of E16 and E17 can be bypassed by issuing the command DIRECT MS in the K-vector of the word containing MSGO and MSRS. In effect, this causes the operation performed to be equivalent to having zero in E16 and all ones in E17. Addressing Main Store beyond installed addresses results in a Program Check with all ones on the MOD bus.

If the MS ADDRESSING AND PROTECTION option is not installed, the effect is that of having DIRECT MS always on. Thus E16 and E17 never participate in Main Store addressing and are available for other use.

The value in FMOD must be established no later than the T-period in which the GATE MS occurs. Both FMIX and FMOD have an extended address function as shown:

Value	FMIX	FMOD
0 - 31	Local Store Registers	Local Store Registers
32 - 39	External Store E0 - E7	External Store E0 - E7
40 - 64	Source of all ones	Null operation

This permits both address and data for Main Store operations to be placed in the External Store Port registers as well as in Local Store.

After READ MAIN STORE has been initiated, any subsequent MSGO or MSRS signals will be ignored until MS BUSY is turned off at the end of the full memory cycle. The minimum timing between Leading Edge of MSGO and Trailing Edge of GATE MS is 640 nanoseconds. The minimum period of successive MSGO signals is 800 nano-seconds.

5.4.3.2 WRITE MAIN STORE

Since the MIX bus is shared for both address and data, a WRITE MAIN STORE operation requires that the data be placed on the MIX bus after the address is established. The single command MSGO is issued as for the READ MAIN STORE operation. At some time thereafer, the value of the data is established by the combination of a changed value in FMIX or the register it references. Then the command MSRS is issued and the memory cycle completes, placing the new data on the MIX bus into the memory at the address previously specified.

In addition to all the requirements for READ MAIN STORE (see Section 5.4.3.1), one must establish FMIX and MIX no later than the T-step before MSRS is issued.

The memory cycle time is variable, depending on when MSRS is issued. In particular, the leading edge of the next MSGO should not occur less than 400 nanoseconds after the Trailing Edge of MSRS or less than 800 nanoseconds after the Leading Edge of the previous MSGO.

5.4.3.3 READ-MODIFY-WRITE

Main Store may be operated in a READ-MODIFY-WRITE mode in order to modify the contents of a memory location based on the value read. This operation is done by starting a WRITE MAIN STORE, waiting for DATA AVAILABLE, extracting the data with GATE MS, and finally issuing MSRS when the modified data is ready on the MIX bus to be written back into memory. Thus the only difference between a normal WRITE MAIN STORE and READ-MODIFY-WRITE is in the timing of the MSRS signal. The timing rules of Sections 5.4.3.1 and 5.4.3.2 apply.

5.4.4 EXTERNAL OPERATIONS

The commands for external operations are treated in this section because, from a programming point of view, they are quite similar to the commands for initiating various memory reference operations.

Two nanoprimitives are available for initiating external operations. These are RIO ("Read IO") and XIO ("Transmit IO").

Execution of the command RIO requires that the port be previously selected by setting the appropriate value in KA . The RIO command causes the following action:

1. The port-RIO pulse is sent immediately (leading edge).
2. The port register (in External Store) is cleared.
3. The 18 incoming data lines are gated into the port register.

Execution of the command XIO also requires that the port be previously selected in KA. If required for the external operation, the data on the G-bus and on the input to the "phantom" register (FIPH) must be established in the same T-Step as the XIO. The XIO command causes an XIO pulse to be sent to the selected port.

Additional information on the external interface is given in Section 4.6 and Section 8.

5.5 DATA TRANSFER FUNCTIONS

5.5.1 T TRANSFERS

All T-Transfers are involved with establishing the primary control state of the machine. Thus such transfers take place on the trailing edge T-Clock. The commands causing T-transfers are:

LOAD NPC - causes a transfer of 10 bits into the NanoProgram Counter (NPC) from one of three sources as specified:
(KN) - 10 bits from the KN field of the active K-Vector.
(SEQUENCE) - current value in NPC plus one modulo 1024.
(CS) - low order 3 bits of FIDX plus high order seven bits of COD bus in the following order:

```
          FIDX                     COD BUS
    -------------  -------------------------------------
   / X X X A B C / E F G H I J D X X X X X X X X X X X /
   --------------  -------------------------------------


           ----------------------- LOAD NPC (CS) also causes the
   NPC    / A B C D E F G H I J /  low order 11 bits from the COD
          -----------------------  bus to be saved in a dedicated
                                   register for use by LOAD R31.
```

LOAD R31 - clears the high order seven bits of R31 and causes a transfer, into the A and B fields of R31, of the low order 11 bits from the COD bus that was saved by the most recent LOAD NPC (CS).

GATE NS UNCONDITIONALLY -causes a transfer of the 360 bits last accessed from nanostore to be gated into the Control Matrix with execution begun in T1 of the word gated, unless T1 is skipped as a result of SKIP issued concurrently with GATE NS UNC.

GATE NS - same as GATE NS UNCONDITIONALLY provided the test specification in the current T-step is satisfied. Otherwise, the command is ignored. See Section 5.7.

SKIP - causes all of the T-vector commands in the next T-step to be ignored if the test secification in the current T-step is satisfied. Otherwise the commands will be executed normally. GATE NS and SKIP are mutually exclusive in a T-step.

5.5.2 F TRANSFERS

Six-bit transfers into and out of F-store (including F increment, F decrement and ALUF operations; see section 5.6.6 and 5.6.7) are controlled by three groups of fields within the active T-vector:

```
     GROUP 0:   [FSEL0] [AUX0] [IN0] [OUT0]
                   5       3     1      1          (bits)

     GROUP 1:   [FSEL1] [AUX1] [IN1] [OUT1]
                   5       3     1      1          (bits)

     GROUP 2:   [FSEL2] [AUX2] [IN2]
                        [AUX3]      [OUT2]
                   5       3      1     1          (bits)
```

Since the IN and OUT controls serve as nanoprimitive commands, six (at most) such transfers may occur in a single T-step. The six commands are defined as follows:

```
IN0:    AUX [AUX0] (with its source-AUX encoding) ---> F [FSEL0]

OUT0:   F [FSEL0] ---> AUX [AUX0] (with its destination-AUX encoding)


IN1:    AUX [AUX1] (with its source-AUX encoding) ---> F [FSEL1]

OUT1:   F [FSEL1] ---> AUX [AUX1] (with its destination-AUX encoding)


IN2:    AUX [AUX2] (with its source-AUX encoding) ---> F [FSEL2]

OUT2:   F [FSEL2] ---> AUX [AUX3] (with its destination-AUX encoding)
```

Normally, only two or three of the F transfers are commanded in a given T-step since the F selection must be common for each IN/OUT transfer. The AUX encodings are shown schematically in Figure 5.5.2A.

ENCODING OF F TRANSFERS   Figure 5.5.2A

```
       GROUP 0                      GROUP 1                      GROUP 2

------------------          ------------------          ------------------
I SOURCE [AUX0]    I        I SOURCE [AUX1]    I        I SOURCE [AUX2]    I
I  0  000   A      I        I  0  000   A      I        I  0  000   A      I
I  1  001   B      I        I  1  001   B      I        I  1  001   B      I
I  2  010   SW     I        I  2  010   C      I        I  2  010   KX     I
I  3  011   KA     I        I  3  011   KA     I        I  3  011   KA     I
I  4  100   KB     I        I  4  100   KT     I        I  4  100   KB     I
I  5  101   G(GSPEC) I      I  5  101   G(GSPEC) I      I  5  101   G(GSPEC) I
I  6  110   ALUF   I        I  6  110   INCF1  I        I  6  110   INCF2  I
I  7  111   IO ID  I        I  7  111   DECF1  I        I  7  111   DECF2  I
------------------          ------------------          ------------------
          I                           I                           I
     IN0  I                      IN1  I                      IN2  I
          V                           V                           V
    ------------                ------------                ------------
    I F (FSEL0) I               I F (FSEL1) I               I F (FSEL2) I
    ------------                ------------                ------------
          I                           I                           I
     OUT0 I                      OUT1 I                      OUT2 I
          V                           V                           V
  ---------------             ---------------             ---------------
  I DEST [AUX0]  I            I DEST [AUX1]  I            I DEST [AUX3]  I
  I  0  000  A   I            I  0  000  A   I            I  0  000  A   I
  I  1  001  B   I            I  1  001  B   I            I  1  001  B   I
  I  2  010  C   I            I  2  010  C   I            I  2  010  C   I
  I  3  011  KA  I            I  3  011  KA  I            I  3  011  KA  I
  I  4  100  KB  I            I  4  100  KB  I            I  4  100  KB  I
  I  5  101  KSHC I           I  5  101  KX  I            I  5  101  KSHC I
  I  6  110  KALC I           I  6  110  KALC I           I  6  110  KT  I
  I  7  111  KS  I            I  7  111  KSHA I           I  7  111  KSHA I
  ---------------             ---------------             ---------------
```

Notes: G(GSPEC) is defined below in this section. Transfers involving
        this AUX source must appear in STRETCHed T-steps.
        INCF1, DECF1, INCF2 and DECF2 are the results of the F increment
        and F decrement. They are described in 5.6.6.
        ALUF is described in 5.6.6. IO ID is described in 4.6.

G SPECIFIER

Listed as a source in F transfers is the quantity G(GSPEC). Rather than being a single source field, G(GSPEC) indicates one of 16 possible sources on the "G-bus", selected by the value of a single four-bit field (GSPEC) in the active T-vector. The normal use of GSPEC is to indirectly specify a six bit value by referencing one of the G registers (the last 12 F registers). However the complete use of GSPEC includes the following six disjoint nanoprimitive operations. The G-specifier (GSPEC) is used:

1. to specify a source for certain 6-bit transfers;
2. to specify ALUF input selection (5.6.7);
3. to specify the operand in all INC MPC operations (5.6.5);
4. as part of the QM-1 external interface (4.6).
5. in the GENERATE/CLEAR INTERRUPT facility (4.5.2.4).
6. to specify the destination of INDEX ALU operations (5.6.4).

Hence, unless a bit pattern in the GSPEC can be shared to advantage, these five types of operations must be considered mutually exclusive in a T-step.

When involved in operations (1), (2), (4), (5) or (6) above, the GSPEC selects one of the 12 G's or one of four "pseudo-G's", as follows:

VALUE OF GSPEC	G REGISTER SELECTED
0	F20=G0
1	F21=G1
2	F22=G2
3	F23=G3
4	F24=G4
5	F25=G5
6	F26=G6
7	F27=G7
8	F28=G8
9	F29=G9
10	F30=G10
11	F31=G11

VALUE OF GSPEC	PSUEDO G SELECTED	COMMAND SPECIFICATION
12	KSHA	"G KSHA"
13	B	"G B"
14	KS	"G KS"
15	KX	"G KX"

The term, G(GSPEC) has been used to refer to any of these 16 selections.

When GSPEC is used in (3) the low order two bits are used directly to specify the INC MPC operand (see section 5.6.5).

5.5.3 R TRANSFERS

All of the transfers to and from Local Store and External Store are classified as R Transfers since they occur on the Trailing Edge R-clock. Actually such transfers are divided into two classes; input and output, as viewed from the store involved. All output transfers are enabled as soon as the bus control is established; input transfers require an explicit command to enable the gating of the data.

Thus output from local store or external store is enabled on all of the following buses as soon as the corresponding F register selects one of the appropriate registers:

BUS	CONTROL	DATA USED WHEN:
MIX	FMIX	MSGO or MSRS
CIA	FCIA	READ CS or WRITE CS
CID	FCID	WRITE CS
EID	FEID	LOAD ES
AIL	FAIL	ALU always operating
AIR	FAIR	" " "
SID	FSID	SHIFTER always operating
INC	FMPC ***	MPC and INDEX facility always operating
EOD	FEOA	GATE ES

Input to local store or external store requires not only a bus control but also a command to cause the actual gating. For the six input buses to local store and the input to external store, these are:

BUS	CONTROL	DATA TRANSFERRED WHEN
MOD	FMOD	GATE MS
COD	FCOD	GATE CS
EOD	FEOD	GATE ES
AOD	FAOD	GATE ALU
SOD	FSOD	GATE SH
INC	FMPC ***	INC MPC or INDEX
EID	FEIA	LOAD ES

Note: ***-INDEX uses another selection mechanism, see section 5.6.4.

The input to all transfers is determined by the state of the controls as they existed following the R-Clock in the T-step in which the transfer is issued. The transfer is completed following the R clock of the next T-step.

5.6 DATA MANIPULATION FUNCTIONS

5.6.1 GENERAL

A variety of data manipulation functions are provided in the QM-1. Section 5.6 treats each of the major components available for modifying data in either the 18-bit or 6-bit domains. These components are:

ARITHMETIC-LOGIC UNIT (ALU)

SHIFTER AND SHIFTER EXTENSION

INDEX ALU

MPC FACILITY

F REGISTER INCREMENT AND DECREMENT FACILITY

ALUF

RMI

The first four operate on data in the 18-bit portion of the machine. The next two are for manipulation of six-bit data. The last component provides optional manipulative capability applied to data arriving from Main Store.

In addition, R31 provides an interface by which 18-bit data may be manipulated in six-bit sub-fields. Finally, an "AUTO OR" capability exists for logically combining two values in either domain with a simultaneous transfer.

All of the data manipulating components described in the following sections may be operated in parallel. They combine to provide an extremely powerful arithmetic and logical capability.

5.6.2 OPERATION OF THE ARITHMETIC-LOGIC UNIT

The Arithmetic-Logic Unit (ALU) output is determined as a function of:

18 bit input on the ALU Input Left (AIL) bus.
18 bit input on the ALU Input Right (AIR) bus.
A single bit, the Carry-In-Hold (CIH) flipflop.
A 6 bit value in the K register (KALC) used to control the ALU.
A single bit in the F register (FIDX) used to specify 16 or 18 bit mode.

The basic function is determined by the low order 4 bits of KALC as follows:

```
         -------------------------------------------------
  KALC   I  bit  I  bit  I  bit  I  bit  I  bit  I  bit  I
 LAYOUT  I   5   I   4   I   3   I   2   I   1   I   0   I
         -------------------------------------------------
          Decimal Logic
          Control Control <------basic ALU function----->
           Code    Code
```

ALU LOGICAL OUTPUT TRUTH TABLE

Left Input	Right Input 0	Right Input 1
0	NOT BIT 3	NOT BIT 2
1	BIT 0	BIT 1

The individual bit outputs are determined by the two input bit values and the 4 low order bits of KALC as indicated in the truth table on the right. For example, the eXclusive OR function results when KALC is 011001. This can be seen by inserting the control bits into the truth table as specified. Thus all of the 16 Boolean functions are available and their encoding can be deduced from the truth table given.

In all of the cases above, the Logic Control Code is set to 1 to disable the internal carry into each bit. When the Logic Control Code is 0, the operation on each bit is modified by the carry into that bit. The Carry Into the bit is eXclusive OR'ed with the result above to produce the modified result for the bit. This is shown in the modified truth table below.

Here, CIN is defined as the carry into the bit in question. For the low order bit, CIN is defined as the value of the Carry In Hold (CIH) flipflop. For each subsequent bit, CIN is the same as the Carry Output from the preceding bit.

The Carry Output from any bit depends on the inputs to the bit (including CIN) and on the function specified as before. This relationship is given in the final truth table below.

ALU CARRY OUTPUT TRUTH TABLE

	Right Input	
Left Input	0	1
0	NOT BIT 3 AND CIN	NOT BIT 2 AND CIN
1	NOT BIT 0 OR CIN	NOT BIT 1 OR CIN

ALU ARITHMETIC OUTPUT TRUTH TABLE

	Right Input	
Left Input	0	1
0	NOT BIT 3 XOR CIN	NOT BIT 2 XOR CIN
1	BIT 0 XOR CIN	BIT 1 XOR CIN

The Carry Output is valid independent of the setting of the Logic Control Code. This is a consequence of the fact that Logical Mode only inhibits the effect of Carry on the OUTPUT BIT and causes no change in Carry generation within the ALU.

This detailed coverage of the ALU has been included so that any question about the ALU operation can be answered. For normal use, the table on the next page suffices for encoding all ALU operations.

Normal operation of the ALU is in 18 Bit Mode. This results when the 16 Bit Mode Control (high order bit in FIDX) is reset. When in 18 Bit Mode, all 18 bits are active on the inputs to the ALU. However, when 16 Bit Mode Control is set, Bit 15 on each of the inputs (AIL and AIR) is replicated to form the inputs to Bit 16 and Bit 17. This applies to all ALU operations except "PASS LEFT" (011111) which provides a means of transmitting all 18 bits through the ALU even when in 16 Bit mode. Otherwise, the ALU operates identically in either 16 or 18 Bit Mode.

ALU control is taken from the KALC field in the active K-vector. The high-order bit of KALC is the DECIMAL control; when it is on, decimal correction words are forced onto the SOD bus to reflect current ALU activity, as per section 4.2.3.1.

The remaining five bits of KALC control ALU function as follows:
(L,R=LEFT,RIGHT; 2's complement convention assumed)

BIT 3 THRU BIT 0 OF KALC	LOGICAL FUNCTIONS KALC BIT 4 = 1	ARITHMETIC FUNCTIONS KALC BIT 4 = 0		Carry-out bit is defined as Carry-in bit .....
		CARRY IN HOLD = 0	CARRY IN HOLD = 1	
0000	NOT L	L - 1	L	OR L
0001	NOT (L AND R)	(L AND R) - 1	L AND R	OR (L AND R)
0010	NOT L OR R	(L AND NOT R) - 1	L AND NOT R	OR (L AND NOT R)
0011	ALL ONES	ALL ONES	ALL ZEROS	OR ZERO
0100	NOT (L OR R)	(L OR NOT R) + L	(L OR NOT R) + L + 1	
0101	NOT R	(L OR NOT R) + (L AND R)	(L OR NOT R) + (L AND R) + 1	
0110	NOT (L XOR R)	L - R - 1	L - R	
0111	L OR NOT R	(L OR NOT R)	(L OR NOT R) + 1	AND (L OR NOT R)
1000	NOT L AND R	(L OR R) + L	(L OR R) + L + 1	
1001	L XOR R	L + R	L + R + 1	
1010	R	(L OR R) + (L AND NOT R)	(L OR R) + (L AND NOT R) + 1	
1011	L OR R	L OR R	(L OR R) + 1	AND (L OR R)
1100	ALL ZEROS	L + L	L + L + 1	
1101	L AND NOT R	L + (L AND R)	L + (L AND R) + 1	
1110	L AND R	L + (L AND NOT R)	L + (L AND NOT R) + 1	
1111	L [PASS LEFT]	L	L + 1	AND L

The output of the ALU consists of the following:

An 18 bit output determined by the above rules.
A carry-out condition (carry out of the high order bit).
An overflow condition determined by the EXclusive OR of the carry
out values from the two high order bits.

The 18-bit output value is used as the 18-bit input to the Shifter Extension discussed in section 5.6.3. This value may be passed directly to the ADD bus or it may be shifted before transfer.

The carry-out condition may be transferred to the Carry-Out-Hold for testing and it may be also transferred to the Carry-In-Hold for future use. Section 4.2.3.4 covers Carry Control. Testing of the Carry Out and Overflow conditions is discussed in Section 5.7.

In general, operation of the ALU requires two T-periods; either one stretched T-step or two unstretched T-steps. Detailed timing of both the ALU and the Shifter is covered in Section 5.6.3.

5.6.3 OPERATION OF THE SHIFTER AND SHIFTER EXTENSION

The Shifter and Shifter Extension form a combined shift matrix unit operating on a total of 36 bits entering from the ALU (high order 18 bits) and from the SID bus. This unit may operate in either Single mode (passing the ALU output on to the ADD bus) or Double mode (shifting the ALU output before it reaches the ADD bus). The 36 bit output to the ADD and SOD buses is determined as a function of:

18 bit input from the ALU.
18 bit input from the SID bus.
A 6 bit value in the K register (KSHA) used to specify the shift amount.
A 6 bit value in the K register (KSHC) used to control the shifter
A single bit, the Carry-Out-Hold (COH) flipflop.
The high order bit (Decimal Control Code) of the ALU control, KALC.

KSHA specifies Shift Amount (number of positions), and is interpreted modulo x, where x is appropriate for the type of shift specified in KSHC.

KSHC is interpreted as follows:

```
          --------------------------------------------------
KSHC      I  BIT  I  BIT  I  BIT  I  BIT  I  BIT  I  BIT  I
LAYOUT    I   5   I   4   I   3   I   2   I   1   I   0   I
          --------------------------------------------------
           LEFT    RIGHT        TYPE          MODE  DIRECTION
          CONTROL CONTROL  00 Circular     0 Single  0 Left
          SWITCH  SWITCH   01 Logical      1 Double  1 Right
                           10 Arithmetic
          (section 4.2.3.4) 11 Undefined
```

TIMING OF ALU AND SHIFTER

Assume that the LAST trailing-edge operation that changes any input relevant to an ALU/shift process is commanded in T-period Tn. Then the earliest T-period (not T-step) in which the data outputs of that process are available as defined for gating out by a trailing-edge nanoprimitive is given by Tn+x, where x depends on the process, as follows (TEST OUTPUTS are also available in Tn+x, and are discussed further in section 5.7.2):

Pass data through ALU (KALC = "PASS LEFT"), then bypass the SHIFTER EXTENSION to ADD (KSHC specifies single shift) ------------------------------------ X = 2.

Pass data through ALU ("PASS LEFT"), then double shift ----------------- X = 2.

ALU operation (not "PASS LEFT"), then bypass the SHIFTER EXTENSION ----- X = 2.

Any single shift, or SID to SOD with no shift (KSHA = 0) --------------- X = 2.

ALU operation (not "PASS LEFT"), then double shift (special case: SID and shift control inputs not needed until Tn+1) ----------------------------- X = 3.

The ALU and Shifter are not pipeline devices; inputs to a process must be held stable for the duration of that process. (A new process is defined to begin when any input changes; at that time, the result of the previous process is considered invalid.) If this rule is not followed, outputs are undefined.

5.6.4 OPERATION OF THE INDEX ALU

A special data manipulation unit, the INDEX ALU, is provided in order to facilitate rapid indexing and logical operations. In particular, this feature of the machine is useful in computing addresses and in masking operations, although it is not restricted to these tasks in any sense. Operation of the INDEX ALU function can be accomplished in a single stretched T step, or in 2 adjacent unstretched T-steps. The result is gated into local store with the nanoprimitive "INDEX" (see 5.2.2), and is testable in the next T-Step as a special test "R INDEX NOT ZERO" (see 5.7)

The implementation of the function requires the dedicated use of the third F transfer mechanism. Namely the following Control Matrix fields are used in any T step in which the function is invoked.

AUX2	Selects local store source register
FSEL2	Selects arithmetic or logical function
AUX3	Selects index register
GSPEC	Selects local store destination register

Except for those fields, the function operates independently and concurrent with all other functions of the QM-1. The illustration below shows the data paths that are involved. Note that a local store register always receives the result of the operation.

```
G(GSPEC)  ---------------------------
--------->I                         I
          V                         I
===================                 I                      ==================
I       28        I                 I                      I      E 8       I
I                 I        ======================          I      thru      I
I      Local      I        I      18   BIT      I          I      E 19      I
I                 I        I                    I          I   ----------   I
I      Store      I        I ARITHMETIC LOGIC   I          I   MOD   BUS    I
I                 I        I                    I          I  -----------   I
I    Registers    I        I       UNIT         I          I   COD   BUS    I
===================        ======================          ==================
          I                 ^       ^      ^                       I
--------->I                 I       I      I                       I<---------
AUX(AUX2) V                 I     FSEL2    I                       V AUX(AUX3)
           ------------------>              <------------------
```

The three register selection controls used in INDEX ALU operation are encoded as follows:

AUX2 - provides an indirect reference to a local store register which will be used as the left input to the INDEX ALU. The field is decoded by the same physical hardware used to decode AUX2 in its primary capacity as an input select for an F register transfer. Therefore the values associated with each value of AUX2 are the same as when it is used to select an AUX to F transfer.

AUX2	FIELD SELECTED
0	A
1	B
2	KX
3	KA
4	KB
5	G(GSPEC)
6	Not Used
7	Not Used

The selected field, A or B or KX etc. provides the local store register source selection. If used, GSPEC must be repeated in the previous T-step.

AUX3 - an indirect reference to one of 16 possible sources of an 18 bit index operand which will be used as the right input to the INDEX ALU. Twelve of these are External registers; two are sources of all ones; the remaining two are the memory buses MOD and COD respectively.

AUX3	SELECTS
0	A
1	B
2	KT
3	KB
4	F28=G8
5	F29=G9
6	F30=G10
7	F31=G11

CONTENTS	SELECT	"X" OPERAND
0	xx0000	E8
1	xx0001	E9
-	------	---
11	xx1011	E19
12	xx1100	ALL ONES
13	xx1101	ALL ONES
14	xx1110	MOD
15	xx1111	COD

GSPEC - an indirect reference to a local store register to receive the result of the INDEX ALU operation. The field is decoded by the same hardware used to decode GSPEC when it is used to select an input for a F register transfer. Thus the field selected by GSPEC is the same as when it is used to select an AJX to F transfer.

GSPEC	FIELD SELECTED
0	F20=G0
-	------
11	F31=G11
12	KSHA
13	B
14	KS
15	KX

The controls providing for selection of a local store register source and destination and for selection of an index operand from either an E register or directly from Main Store or Control Store have been described. There remains the manner in which the specific function is selected by the FSEL2 field.

FSEL2 - either a direct selection of one of 13 arithmetic and logical functions, or an indirect specification of one of 18 possible 6 bit fields containing one of the standard 48 function codes. ("L" refers to a Local Store register; "X" refers to an External Store register, MOD bus, or COD bus.

FSEL2	FUNCTION SPECIFIED
00	L - 1 -----> L
01	L + 1 -----> L
02	L XOR X ---> L
03	ALL ONES --> L
04	ALL ZERO --> L
05	NOT X -----> L
06	L - X -----> L
07	L AND X ---> L
08	L OR X ---> L
09	L + X -----> L
10	X ---------> L
11	NOT L -----> L

FSEL2	FUNCTION SPECIFIED BY	OR FUNCTION SPECIFIED
12	A	
13	B	
14	KA	
15	KB	
16	F16 - FMPC	
17	F17 - FIDX	
18		L --> L
19		L --> L
20	F20 - G0	
--	--- --	
31	F31 - G11	

FUNCTION SPECIFICATION

FUNCTION CODE BIT 3 2 1 0	BIT 5 4: * 1 (*=DON'T CARE)	5 4: 0 0	5 4: 1 0
0 0 0 0	NOT L	L - 1	L
0 0 0 1	NOT (L AND X)	(L AND X) - 1	L AND X
0 0 1 0	NOT L OR X	(L AND NOT X) - 1	L AND NOT X
0 0 1 1	ALL ONES	- 1 (2's complement)	ALL ZEROS
0 1 0 0	NOT (L OR X)	L +(L OR NOT X)	L +(L OR NOT X)+ 1
0 1 0 1	NOT X	(L AND X) + (L OR NOT X)	(L AND X)+(L OR NOT X)+1
0 1 1 0	NOT(L XOR X)	L - X - 1	L - X
0 1 1 1	L OR NOT X	L OR NOT X	(L OR NOT X) + 1
1 0 0 0	NOT L AND X	L + (L OR X)	L + (L OR X) + 1
1 0 0 1	L XOR X	L + X	L + X + 1
1 0 1 0	X	(L AND NOT X)+(L OR X)	(L AND NOT X)+(L OR X)+1
1 0 1 1	L OR X	L OR X	(L OR X) + 1
1 1 0 0	ALL ZERO	L + L	L + L + 1
1 1 0 1	L AND NOT X	(L AND X) + L	(L AND X) + L + 1
1 1 1 0	L AND X	(L AND NOT X) + L	(L AND NOT X)+ L + 1
1 1 1 1	L	L	L + 1

5.6.5 MPC OPERATIONS

Four Local Store registers (R24, R25, R26 and R27) have a special increment capability to facilitate their use as Micro Program Counters. The register currently designated as the MPC is determined by the contents of FMPC (mod 4).

The current content of the designated MPC is continuously being added to the following four values:

+1
+2
B (low order 6 bits of R31, sign extended, two's complement addition).
AB (low order 11 bits of R31, sign extended, two's complement addition).

The results of these computations may be used as a Control Store address for READ CS or WRITE CS, or they may be gated back into the designated MPC, using the command INC MPC. The CS ADDR SELECT field is used to select the desired value for Control Store operations. The GSPEC field is used to select the desired result in INC MPC operations. The encodings are as follows:

MPC RESULT	CS ADDR SELECT	GSPEC VALUE
MPC	010	(X=ignored)
MPC+1	011	XX00
MPC+2	100	XX01
MPC+B	101	XX10
MPC+AB	110	XX11

All MPC operations involve 18 bit arithmetic; the high order bit is the ROCS indicator bit. Thus incrementing 2**17 - 1 by 1 produces a "negative" which actually addresses ROCS(0).

The INC MPC Nanoprimitive is a trailing-edge operation; the incremented value is loaded into the appropriate Local Store MPC (the one selected by FMPC) at the end of the T-step in which the nanoprimitive is executed. The propagation time through the adding circuits is such that if FMPC, MPC, A or AB is changed by any trailing-edge command in the T-step prior to the one containing the INC MPC command, the T-step containing the INC MPC command must be STRETCHed.

5.6.6 INCREMENT F AND DECREMENT F OPERATIONS

A facility is available for incrementing or decrementing the contents of any F register. This is acomplished by selecting the desired F register for an AUX to F transfer, selecting INCF or DECF as the AUX source and enabling the transfer with the IN command. Since INCF and DECF are available as AUX sources only in Group 1 and Group 2, at most two F-registers may be loaded with (either) an incremented or decremented value in a T-step.

The F register increment and decrement facility operates by continuously performing an increment and decrement operation on the contents of the F registers selected by the FSEL1 and FSEL2 fields. Four values are available as outputs:

F(FSEL1) + 1   F(FSEL1) - 1   F(FSEL2) + 1   F(FSEL2) - 1

These values are selected by the corresponding AUX1 and AUX2 fields. The commands IN1 and IN2 cause the selected value to be gated back into the F register as a trailing-edge F transfer. The F register increment and decrement are unsigned six-bit operations. For example, incrementing 111111 produces 000000. The input to INCF/DECF(FSEL1) is available for testing with the F NOT ZERO test (see section 5.7.2.), where True indicates a non-zero condition of F(FSEL1) at the LE of this T-step.

Because of the propagation time of the adding circuits, an INCF or DECF must be placed in a STRETCHed T-step in order to be completely self-contained. However, it may be successfully placed in an unstretched T-step provided the following two conditions are met:

1) The value in the F register is not changed by an F transfer in the preceding T-step.

2) The same F register selection (FSEL1 or FSEL2) is specified in the preceding T-step.

Note: Incrementing or Decrementing FIPH should be avoided since this connects an adder's output directly to one of its inputs and results in an unstable condition.

5.6.7 ALUF OPERATIONS

A six-bit arithmetic logical facility, designated "ALUF", is available as a QM-1 option. This facility provides full arithmetic and logical operations, using the contents of two F registers as the two operands, with the result placed in any desired F register.

The ALUF output is transferred into an F-register by the normal AUX to F trailing-edge nanoprimitives that load source AUX fields into an F-register; these controls are presented in section 5.5.2. This transfer is only available in Group 0. FSELO determines the F register receiving the result.

The left data input is the contents of the F register selected by FSEL1. This is designated "F" in the table below. The right data input is from the F register selected by FSEL2. This operand is referrenced as "R" in Table 5.6.7B. Finally, the arithmetic or logical operation to be performed on these two operands is indirectly determined by the contents of AUX3 as encoded in Table 5.6.7A and Table 5.6.7B.

TABLE 5.6.7A FUNCTION Register Selected by AUX3

AUX3	SELECTS
0	A
1	B
2	KT
3	KB
4	F28=G8
5	F29=G9
6	F30=G10
7	F31=G11

TABLE 5.6.7B ALUF FUNCTION ENCODING AND OUTPUTS

BITS 5 - 0	FUNCTION
XX0000	NOT F
XX0001	NOT (F OR R)
XX0010	(NOT F) AND R
XX0011	ALL ZEROS
XX0100	NOT (F AND R)
XX0101	F XOR R
XX0110	F MINUS R
XX0111	F AND NOT R
XX1000	(NOT F) OR R
XX1001	F PLUS R
XX1010	R
XX1011	F AND R
XX1100	F PLUS F
XX1101	ALL ONES ****
XX1110	F OR R
XX1111	F

Notes: 2's Complement Arithmetic assumed.
****** Default condition when ALUF Option is not installed.
If one of the F inputs selects FIPH, then the input to the ALUF may originate from the AUX field transmitted to FIPH.

The ALUF unit is continuously operating, using the inputs and function as specified by the FSEL1, FSEL2, and AUX3 fields at the leading edge of each T-Step. The output will be gated as a trailing edge AUX to F transfer only when AUX0 specifies ALUF and the IN0 control specifies an AUX to F transfer in Group 0. Any T-step using the ALUF output must be STRETCHed in order for the output to be stable, within one T-Step. An ALUF operation may also be specified across two unstretched T-Steps as long as all inputs remain stable. ALUF and IN0 would only be specified in the second T-Step.

If the ALUF Option is not installed, a source of all ones is available by selecting the appropriate ALUF controls.

5.6.8 RMI OPERATIONS

A special unit, designated the RMI unit, is available as a QM-1 option, for manipulating the data read from Main Store on the way into Local Store or External Store. This unit permits selection among three sets of dynamically variable ROTATE, MASK and INDEX operations on the Main Store data. It is ideally suited for emulation of various main store machines having a variety of memory widths and formats.

A preliminary description of the RMI unit is given is section 4.2.6.4. Further specifications are available on request.

5.7 CONDITIONAL FUNCTIONS

5.7.1 TEST FIELDS AND MASKS

Three classes of test conditions are available for decision making at the nano-program level. These are:

Local conditions	- six current output conditions available from the ALU and Shifter.
Global conditions	- six output conditions from the ALU and Shifter that were previously saved in F register FIST.
Special conditions	- a set of six special machine status conditions that available for testing.

Within any class, one or more of the conditions may be simultaneously tested for being present or absent. An individual six-bit mask field in the active K-vector is provided for testing each class. The bits set in the mask select the conditions to be tested.

KT masks the local conditions.
KS masks the global conditions (FIST);
KX masks the special conditions.

The arithmetic test fields and masks (KS, KT, and FIST) are formatted as shown below. (The conditions are defined in section 4.2.3.5.)

Shifter High Bit	ALU Carry	ALU Sign	ALU Result	Combined Overflow	Shifter Low Bit
Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

The special test field and mask (KX) is formatted as follows:

	Not used Reserved	PROGRAM CHECK	R INDEX NOT ZERO	MS BUSY	MS DATA INVALID	F NOT ZERO
	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
See Section:		4.5.2.2	5.6.4		4.2.6	5.6.6

The remainder of the testing facility consists of two fields in the active T-Vector, a one-bit ACTION selector and a three bit TEST SPECifier.

```
     ACTION        0 - SKIP (inhibit all controls in next T-step).
                   1 - GATE NS (load control matrix with nanoword last
                            accessed from nanostore).

     TEST 000 (=0) - Do not execute ACTION.
          001 (=1) - Execute  ACTION unconditionally.
```

Interpretation of other encodings is "EXECUTE ACTION IF:"

```
          010 (=2) - FIST AND KS EQUAL ZERO
          011 (=3) - FIST AND KS NOT EQUAL ZERO
          100 (=4) - LOCAL CONDS AND KT EQUAL ZERO
          101 (=5) - LOCAL CONDS AND KT NOT EQUAL ZERO
          110 (=6) - SPECIAL CONDS AND KX EQUAL ZERO
          111 (=7) - SPECIAL CONDS AND KX NOT EQUAL ZERO
```

(A test condition is "1" if true, "0" if false; the unused position in KX matches against "0".)

Hence at most one class of conditions may be tested in a T-step. However, it is possible to take action upon combinations of conditions within a class; for example, a useful test involves specifying both the S and R bits in KT, thus testing for a full 18-bit or 16-bit arithmetic result.

5.7.2 TESTS

The execution of an action (SKIP or GATE NS) as the result of a test is considered a trailing-edge event. However, the timing of conditional action is such that all T-steps containing a conditional SKIP or GATE NS must be STRETCHed. In addition, the test condition must be stable at the time the test is made. This is assured by the rules below.

The local test conditions are stable at the time the output of the Shifter or ALU is stable for gating into Local Store. These times are covered in Section 5.6.3. Conditional ACTION based on the local test conditions may occur in the T-step immediately following the one in which the trailing-edge functions are valid. Alternatively, conditional ACTION is valid in a STRETCHed T-step when the gate functions would have been valid in the same unSTRETCHed T-step.

The global test conditions are the six bits of FIST. This register can be loaded with an ordinary six-bit transfer operation, or as follows:

If the "ALU STATUS ENABLE" bit is on in the active K-vector, then when GATE ALU is executed:

a) The local conditions S, R, and O are loaded into ther counterparts in FIST (trailing-edge).
b) The "C" bit in FIST is loaded as follows (trailing-edge):
   1. If a nanoprimitive is simultaneously executed to load COH, the value loaded into COH is also loaded into FIST-C.
   2. If no such operation is simultaneously commanded, FIST-C is loaded from the current value of COH (the current local CARRY condition).

If the "SH STATUS ENABLE" bit is on in the active K-vector, the two local conditions SHB and SLB are loaded into the two corresponding bits of FIST when GATE SH is executed (trailing-edge).

Conditional Action based on the Global test conditions is valid in the T-step immediately following a T-step in which the state of FIST is changed.

The special conditions MS BUSY and MS DATA INVALID are syncronized by the machine clock. Thus they may be tested in any STRETCHed T-step. Section 5.4.3 covers Main Store timing.

The special condition F NOT ZERO reflects the state of the F register specified by FSEL1 at the leading edge of the testing T-step. The special condition R INDEX NOT ZERO reflects the result of the last INDEX ALU operation prior to the leading edge of the testing T-step. The special condition PROGRAM CHECK reflects the current combined state of all the PROGRAM CHECK conditions. (See section 4.5.2.2).

Note: Unconditional SKIP, GATE NS are simple trailing-edge operations. They do not require that the T-step be STRETCHed.

5.8 MISCELLANEOUS FUNCTIONS

5.8.1 GENERATE INTERRUPT

In order to facilitate using the External Interrupt structure from within a nanoprogram, the GENERATE INTERRUPT command is provided. This command allows the nanoprogrammer to generate or clear any interrupt internally.

The value of G(GSPEC) in T-period 1 selects the interrupt level to be latched or unlatched when the GENERATE INTERRUPT K-bit is set (see section 4.5.2.4).

When the high-order bit of G(GSPEC) is 1, the low-order five bits of G(GSPEC) select the interrupt level (2-31) to be cleared (values 0 and 1 specify no action).

When the high-order bit of G(GSPEC) is 0, the low-order five bits of G(GSPEC) select the interrupt level (2-31) on which to generate an interrupt (values 0 and 1 specify no action).

5.8.2 AUXILIARY ACTION

The AUXILIARY ACTION nanoprimitive uses the value found in F register FACT to command special QM-1 control functions. In general, these auxiliary commands will enable or disable various interrupt and control facilities.
Currently defined AUXILIARY ACTION commands are:

FACT (octal)	Function
00	No Operation. (All undefined commands are also treated as No Operations.)
77	DISABLE interrupt levels 2-31. Overrides the ALLOW INTERRUPT Bits in the active, and all subsequent, K-Vectors, until rescinded by the Enable Interrupt command below.
76	ENABLE interrupts. Rescinds the action of the DISABLE Interrupt command above, and restores the control of interrupts to the ALLOW INTERRUPT bits. Note: Only those interrupt levels masked "on" by the Interrupt Enable Bits in External Store registers 18 and 19 are enabled.
75	SET RELATIVE MS- All Main Store operations not having DIRECT MS on in the K-Vector are relative to MS Base Register and are checked against MS Field Length.
74	SET DIRECT MS - All Main Store operations access Main Store as though DIRECT MS were on in all K-Vectors executed.
64 - 60	Special CS Address Translation Actions. (see APPENDIX B)
57	LOAD ROTATE value - loads Rotate parameter of RMI(UNIT) from the COD bus output.
56	LOAD MASK value - loads Mask parameter of RMI(UNIT) from the COD bus output.
55	LOAD INDEX value - loads Index parameter of RMI(UNIT) from the COD bus output.
40	PROGRAM STOP - if Program Step switch is on this Aux Action will halt the QM-1 as described in section 5.8.3.

Both the IO Interrupt Disable and Enable commands become effective within 2 T-periods after execution of the T-step containing the AUXILIARY ACTION nanoprimitive. If the nanoword containing this command allows interrupts, it must not execute a READ NS using other than the priority branch address during these two T-Periods. The result of a premature READ NS is undefined.

5.8.3 MISCELLANEOUS CONTROL FUNCTIONS

The following external control buttons and switches apply to the QM-1 CPU:

MASTER CLEAR - push button for setting the initial conditions for machine start-up.

START - push button for initiating CPU operation

SINGLE/RUN/DOUBLE - a three position switch which selects either a SINGLE T-step, DOUBLE T-step, or continuous operation mode when the START button is depressed.

MICRO-STEP - a two position switch that causes continuous operation to be stopped following each execution of the LOAD R31 command.

PROGRAM STEP SWITCH - a two position switch which allows an AUX ACTION 40 in a T-Step to stop the T-Clock after 2 T-periods and one T-Clock since the TE of the T-Step containing the AUX ACTION. When this switch is off any AUX ACTION 40 command is ignored.

5.9 REFERENCE LAYOUT AND MAPS

5.9.1 CPU REGISTER ASSIGNMENT AND LAYOUT

LOCAL STORE		EXTERNAL STORE			F STORE	
R0		E0	PORT 0		F0	FMIX
R1		E1	PORT 1		F1	FMOD
R2		E2	PORT 2		F2	FCIA
R3		E3	PORT 3		F3	FAIL
R4		E4	PORT 4		F4	FCID
R5		E5	PORT 5		F5	FAIR
R6		E6	PORT 6		F6	FCOD
R7		E7	PORT 7		F7	FAOD
R8		E8	INDEX 0		F8	FSID
R9		E9	INDEX 1		F9	FSOD
R10		E10	INDEX 2		F10	FEID
R11		E11	INDEX 3		F11	FEOD
R12		E12	INDEX 4		F12	FEIA
R13		E13	INDEX 5		F13	FEOA
R14		E14	INDEX 6		F14	FACT
R15		E15	INDEX 7		F15	FLIV
R16		E16	INDEX 8,	MS BASE ADR.	F16	FMPC
R17		E17	INDEX 9,	MS FIELD LENGTH	F17	FIDX
R18		E18	INDEX 10,	INTERRUPT ENA.	F18	FIST
R19		E19	INDEX 11,	BITS	F19	FIPH
R20		E20		INTERRUPT PEND.	F20	G0
R21		E21		FLAGS	F21	G1
R22		E22	INT ADR (2-4)		F22	G2
R23		E23	(5-7)		F23	G3
R24	MPC REG	E24	(8-10)		F24	G4
R25	MPC REG	E25	(11-13)		F25	G5
R26	MPC REG	E26	(14-16)		F26	G6
R27	MPC REG	E27	(17-19)		F27	G7
R28		E28	(20-22)		F28	G8
R29		E29	(23-25)		F29	G9
R30		E30	(26-28)		F30	G10
MIR R31	"C", "A", "B"	E31	(29-31)		F31	G11

The indicator lamp layout for all CPU registers is show in Figure 5.9.1A.

```
          LOCAL STORE                 EXTERNAL STORE                        INTERRUPTS

R R R R R R R R R R R R R R R R     E REGISTERS    =================        0 1 2
0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3     24/25/26/27    I CPU INDICATOR I        2 2 2
0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0     17<-Bits->0    I  LAMP LAYOUT  I        ^ ^ ^
         Bits 0 - 5                                I Figure 5.9.1A I        I I I
R R R R R R R R R R R R R R   R     E REGISTERS    =================        V V V
0 0 0 0 0 1 1 1 1 1 2 2 2 2   3     28/29/30/31                             1 2 3
1 3 5 7 9 1 3 5 7 9 1 3 5 7 9 1     17<-Bits->0                             1 1 1

                                                                F STORE
R R R R R R R R R R R R R R R R     E REGISTERS    F F F F F F F F F F F F F F F F
0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3     16/17/18/19    0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3
0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0     17<-Bits->0    0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0
         Bits 6 - 11                                           Bits 0 - 5
R R R R R R R R R R R R R R   R     E REGISTERS    F F F F F F F F F F F F F F F F
0 0 0 0 0 1 1 1 1 1 2 2 2 2   3     20/21/22/23    0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3
1 3 5 7 9 1 3 5 7 9 1 3 5 7 9 1     17<-Bits->0    1 3 5 7 9 1 3 5 7 9 1 3 5 7 9 1

                                                       SW   K REGISTERS
R R R R R R R R R R R R R R R R     E REGISTERS  CIH   0     K S S K       GEN   C
0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3     08/08/10/11        1     A H H X       INT   S
0 2 4 6 8 0 2 4 6 8 0 2 4 6 8 0     17<-Bits->0        2       C A
         Bits 12 - 17                                       Bits 0 - 5           A
R R R R R R R R R R R R R R   R     E REGISTERS        3     K A K K       SKP   D
0 0 0 0 0 1 1 1 1 1 2 2 2 2   3     12/13/14/15        4     B L S T             D
1 3 5 7 9 1 3 5 7 9 1 3 5 7 9 1     17<-Bits->0  COH   5       C                 R

                                                              T GENERATOR
                K   N   N           E REGISTERS    A  M  R   0 0 1 2 3 4 4 5 6  T1
                N   P   S           00/01/02/03    U  I  U   0 8 6 4 2 0 8 6 4  T2
                    C               17<-Bits->0    X  S  N                      T3
                R       A                             C      <--Bits 0 - 71-->  T4
                E   R   D           E REGISTERS    A
                G   E   D           04/05/06/07    C  K      0 1 2 3 3 4 5 6 7
                    G   R           17<-Bits->0    T  '      7 5 3 1 9 7 5 3 1
                                                      S
```

5.9.2 NANOSTORE MAP

K SEGMENT LAYOUT

```
BYTE           17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00  BITS
               -------------------------------------------------------
 00            ILE/BR/AN/AM/DA/H1/SS/AS/            KN               I 17-00
               I-----------------------------------------------------I
 01            ISM/GI/AL/**/**/H2/        KA        /        KB      I 35-18
               I-----------------------------------------------------I
 02            I      KSHC       /       KALC       /       KSHA     I 53-36
               I-----------------------------------------------------I
 03            I       KS        /        KX        /        KT      I 71-54
               -------------------------------------------------------
```

LE	LEGAL MICRO ENTRY	H1	HOLD	AS	ALU STATUS ENABLE
SM	SUPERVISOR	H2	HOLD 2	SS	SH STATUS ENABLE
BR	BRANCH	AM	ALLOW MICRO INTERRUPT	GI	GENERATE INTERRUPT
AL	ALTERNATE	AN	ALLOW NANO INTERRUPT	DA	DIRECT MS ACCESS

T SEGMENT LAYOUT

```
T1 T2 T3 T4    17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00  BITS
               -------------------------------------------------------
04 08 12 16    IMG/MR/GM/ RMI /RN/WN/IM/IO/  AUX0   /00/    FSEL0    I 17-00
               I-----------------------------------------------------I
05 09 13 17    IXI/RI/LE/GE/TX/GU/31/GS/I1/  AUX1   /01/    FSEL1    I 35-18
               I-----------------------------------------------------I
06 10 14 18    ICARR CTL/TEST SPC/TA/GA/I2/  AUX2   /02/    FSEL2    I 53-36
               I-----------------------------------------------------I
07 11 15 19    IRC/WC/GC/CS A SEL/LNPC /ST/  AUX3   /AA/IN/   GSPEC  I 71-54
               -------------------------------------------------------
```

MG - MSGO	RC - READ CS	LE - LOAD ES	CARR CTL - CARRY CTL
MR - MSRS	WC - WRITE CS	GE - GATE ES	TEST SPEC- TEST SPECIFIER
GM - GATE MS	GC - GATE CS	GS - GATE SH	CS A SEL - CS ADDR SELECT
RMI- RMI SELECT	IM - INC MPC	GA - GATE ALU	TX - T STOP ; LNPC - LOAD NPC
RN - READ NS	XI - XIO	31 - LOAD R31	GU - GATE NS UNCONDITIONLLY
WN - WRITE NS	RI - RIO	TA - TEST ACTION	I0, I1, I2 - IN0, IN1, IN2
ST - STRETCH	IN - INDEX	AA - AUX ACTION	O0, O1, O2 - OUT0,OUT1,OUT2

6 NANOPROGRAMMING LANGUAGE SPECIFICATION

6.1 GENERAL

Nanoprogramming is the process of defining QM-1 hardware control sequences and implementing their definitions by programming the contents of words in QM-1 Nanostore. A nanoprogram, or logically complete control sequence, can be invoked from one of three sources: Machine Start, interrupt entry, or microinstruction entry. Since it is plausible to regard microinstruction control as the "typical" mode of QM-1 operation, most nanoprogramming can be considered to be the process of defining microinstructions and implementing their definitions by programming the appropriate nanoprimitives in sequences of Nanostore words. Such a sequence of nanowords is called the "nanoprogram" corresponding to the defined "microinstruction".

While nanoprogramming is the most elementary level of programming possible in the QM-1 and has many unique characteristics dependent on QM-1 hardware, it has much in common with any type of programming. In particular, it shares the need for a symbolic language to relieve the programmer of having to remember the details of actual bit locations and absolute codes. This section defines a nanoprogramming language to meet the needs of the nanoprogrammer, much as an assembly language meets the needs of programmers on a more conventional computer.

Although some implementation standards are described, the intent of this section is to present a language specification only, generally avoiding elements (such as assembler directives) that more properly belong in an assembler user's manual.

We would like to acknowledge the efforts of Dr. Bob Nash, at the Department of Computer Science, State University of New York at Buffalo. The Nano-Assembler is based on his definition of the "Nanocode Symbolic Assembler", developed at the university.

[Notes appearing within this chapter, enclosed in brackets, denote temporary restrictions or features found only in the basic Nano-Assembler (Version 1, Level 2).]

6.2 ELEMENTS

6.2.1 SOURCE STATEMENTS

Source statements consist of single records that are either:

a) Comment Statements
b) Label Statements
c) Command Statements
d) Control Statements (briefly discussed in this document)

Each type of statement is complete on a single record. There is no provision for continuation of a statement.

Sets of command statements actually define nanowords. These sets are preceded by a label statement to locate and name the nanoword so defined.

Comment statements are used only to annotate the listing and are otherwise ignored. Control statements provide information and direction for the processor that translates the nanoprograms into absolute bit strings in a form suitable for loading and execution on the QM-1.

All statements, excluding comments, may be subdivided into fields. Any number of fields may occur on label, command, and control statements; as warranted by their immediate applications. Fields are separated by field delimiters, as described below (6.2.4.1).

Each type of statement will be defined in more detail in subsequent sections.

6.2.2 CHARACTER SET

The character set available for writing nanoprograms is a subset of both ASCII and the IBM 029 keypunch. It includes the following characters:

A, B, ..., Z	upper case alphabet
0, 1, ..., 9	decimal digits
" "	blank
.	period
,	comma
:	colon
=	equal

>	greater-than-sign
+	plus
-	minus
/	slash
*	star
"	quotation mark

This character set will be expanded as the ne [illegible] or [illegible] t [illegible]) [illegible] ¬

6.2.3 SYMBOLIC NAMES

Symbolic names are strings of letters, digits, periods, and single occurrences of the blank character preceded by any non-blank character. A symbolic name may begin with a letter or period. Leading blanks are ignored. These strings may be of any length, but only the first 10 characters are used for recognition of symbolic names. If the 10-th character of a name is a blank it is also ignored.

6.2.4 DELIMITERS

6.2.4.1 FIELD DELIMITERS

Label, Command, and Control statements may, optionally, be divided into fields. Two field delimiters are defined. The comma (,) is most frequently used to separate fields, and is treated as a field delimiter on all statement types. It is ignored within comment fields (6.2.5.2) and on comment statements. The blank ( ) may also be used as a field delimiter, but only when two or more blanks immediately follow a legal symbolic name; and where the following field begins with a symbolic name. The reasons for this alternate delimiter are discussed below, in the section on Pseudo Commands (6.3.2.1).

[Version 1, Level 2 restriction; any ocurrence of two or more blanks following a symbolic name will be treated as a field delimiter, regardless of the first component of the following field.]

6.2.4.2 SYMBOLIC NAME DELIMITERS (OPERATORS)

Symbolic names may be delimited by either field delimiters (6.2.4.1) or operators. The basic arithmetic operators + (addition), - (subtraction), * (multiplication), / (division), and = (assignment) are recognized only within command or control fields where aritnmetic expressions are legal. One additional operator is defined, and consists of the character pair "->" (transmit). This operator may be used only where data transfer commands are legal. Finally, the quotation mark (") may follow a symbolic name acting as both a comment field delimiter (see 6.2.5.2 below) and a symbolic name delimiter.

6.2.5 COMMENTS

6.2.5.1 COMMENT STATEMENTS

Any statement that has * in column one of the statement will be treated as a comment statement. It will be printed on the source listing but will have no other effect on the translation.

6.2.5.2 COMMENT FIELDS

Comments may be included within fields on label, command, and control statements by simply enclosing the comment between a pair of quotation marks ("). Comment fields may be placed before or after symbolic names. A comment field that is in effect is terminated upon encountering the end of a statement. A new comment field must be declared on the next statement, in order to continue that comment.

6.2.6 BLANKS

Aside from their use in symbolic names (6.2.3) and as field delimiters (6.2.4.1) strings of blanks are ignored.

6.3 NANOWORD DEFINITION

6.3.1 LABEL STATEMENT

A label statement is defined as a statement that contains either a symbolic or null label declaration.

A symbolic label is indicated by a field containing a symbolic name followed by a colon.

A null label consists of a field containing only a colon, with no preceding characters.

The colon terminates the label field, but not the label statement. Additional fields containing Pseudo Commands (see section 6.3.2.1) may be included on the statement.

A symbolic label is used specifically to pass the symbolic name, and corresponding nanostore location, of a micro-instruction to the Micro-Assembler (described in another document); for use as an actual operation code.

[In Version 1, Level 2, this label cannot be used for reference by any other nanoword; instead a variable symbol name may be equivalenced to the same nanostore location (see example below).]

The occurrence of a label field indicates the end of the preceding nanoword (if any) and begins a new word definition.

Examples of label statements:

1. Beginning of an ADD instruction.
```
   ADD:  "MICRO INSTRUCTION FORMAT ATTRIBUTES GO HERE"
```

2. Complex operation code name and attributes.
```
   DECODE X.R:  MICRO = ABSOLUTE + AB RELATIVE + WORD 2
         ("MICRO" is explained in section 6.3.2.1)
```

3. Variable name used for reference to this nanoword; through KN fields of other nanowords. The special symbolic name "N." is used to access the current nanoword address (see section 6.3.2.1).
```
   ADD:  ADD.OPR = N.,  MICRO = A B ABSOLUTE.
```

[Version 1, Level 2, may use only the method in example 3 to reference other nanowords.]

6.3.2 COMMAND STATEMENTS

Command statements serve to actually define the nanoprimitives desired in a nanoword, and to show in which T-Vector they should appear. Each statement consists of one or more command fields, which are order independent within each T-Vector [not so in Version 1, Level 2; see section 6.3.2.4].

There are three classes of commands: Pseudo Command Operators, Nanoprimitive Commands, and 6 bit Data Transfer Commands. Pseudo Command Operators are used to declare assembly time functions and attributes, affecting the nanoword currently being assembled. Nanoprimitive Commands each define values for up to two K-Vector or T-Vector fields. These commands explicitly identify nanoword fields to be included in the generated nanoprogram, binary output file. The 6 bit Data Transfer Commands are effectively macroscopic nano-primitive commands, and define nanoword fields affecting transfers between 6 bit AUX fields and F registers. A single transfer command may implicitly define up to 5 nanoword fields. All three command classes are discussed in more detail in the following sections.

6.3.2.1 PSEUDO COMMAND OPERATORS

Pseudo Command Operators are required to specify assembly time attributes of the nanoword currently being processed. These attributes include external micro-indtruction format indicators, for use by the Micro-Assembler, and selection of the appropriate T-Vectors to receive specific nanoprimitive commands. There are three types of pseudo commands supported in the Version 1 Nano-Assembler: Nanostore location counter, Micro-instruction attributes, and position declarations.

All pseudo command operators may be delimited by a comma, or at least two blank characters (as described in section 6.2.4.1 above). Use of blanks as the delimiter allows pseudo commands to take on the appearance of operators, as found in more conventional assemblers, where blanks denote separation between operator and parameter fields.

N A N O S T O R E   L O C A T I O N   C O U N T E R

The Nanostore location counter may be accessed, or modified, using arithmetic expressions. This capability permits the assignment of the current nano-location-counter to another symbolic name. This symbol may then be referenced through KN fields of other nanowords, permitting symbolic nano-branch declarations. Continuation nanowords, in the larger nanoprograms, need not have symbolic labels (are not known to the Micro-Assembler) and must therefore be referenced through this alternate means.

```
:  SEARCH.2 = N.   "LABEL STATEMENT WITH NULL LABEL FIELD"
```

It is legal to modify the nano-location-counter, as a method of altering the normally sequential order of code generation. Care must be taken when using this method since the value of N. must be set to one less than that of the next desired nanoword location.

```
"MAKE THE ADD INSTRUCTION MICRO-OPERATION CODE 40"  N. = 40-1
ADD:  ADD LOC = N.   "ADD LOC = LOCATION OF ADD"
```

[The basic Version 1 assemblers have certain predefined variable symbol names, to simplify declaration of functional unit actions (such as ADD, SUB, OVERFLOW, RESULT, etc.). The temporary differentiation between symbolic labels and symbolic names permits use of alternate nanoword reference names, where the predefined symbolic name value must be retained.]

M I C R O - I N S T R U C T I O N   ATTRIBUTE DECLARATIONS

Only a limited set of micro-instruction formats are predefined within the Version 1 Micro-Assembler. Selection of the format to be used by that assembler, when encountering specific micro-instruction operation codes, is provided by the "MICRO" pseudo command at nano-assembly time. Selections may, optionally, be made during micro-assemblies.

Version 1 format selection is provided by a format index number, passed between the two assemblers. The actual format declaration should be made on the label statement, following the symbolic label field. The "MICRO" pseudo command appears as follows:

```
MICRO = <expression>
```

Where <expression> may be any legal arithmetic expression (as discussed in section 6.3.2.2). The list below describes some of the formats supported by the Version 1 Micro-Assembler. A zero valued MICRO declaration (or no declaration) causes the Micro-Assembler to select a default format.

MICRO VALUE --(octal)--	FORMAT ATTRIBUTES
200	"OP M,N" 18 bit, with absolute (*) instruction parameters. M is the 5 bit A field, N is the 6 bit B field.
201	"OP M,N" 18 bit, with parameter M as a 5 bit absolute A field. N is the 6 bit B field, with a micro-location-counter relative value.
207	"OP MN" 18 bit, with parameter MN representing an 11 bit, micro-location-counter relative value.
300	"OP M,V,N" 36 bit, with M representing the 5 bit A field (absolute). V represents the 18 bit signed value of the second word of the instruction. N is the 6 bit B field (absolute).
303	"OP MN,V" 36 bit, with MN representing an 11 bit absolute value (AB field). V represents the 18 bit signed value of the second word.

* In the above table; location-counter relative values are all signed, two's complement, with leftmost bit indicating the sign. The term "absolute" refers to non-relocatable address expressions, as relocatable expressions are illegal within those parameters.

P O S I T I O N D E C L A R A T I O N C O M M A N D S

The Position Declaration Commands determine whether the commands that follow specify K-Vector or T-Vector fields, and in the case of T-Vector specifications also select the T-Vector position. A position declaration must appear prior to any T-Vector commands. K-Vector commands may appear anywhere following the label statement, although it is recommended that they be placed following the K-Vector position declaration, to avoid possible programming errors. Each position declaration consists of a single symbolic name, which may be terminated by either a comma or multiple blank delimiter (see section 6.2.4.1, Field Delimiters). The allowed forms of these declarations are:

....	Only K commands allowed in statement.
X...	All T commands apply to T-vector 1 of the current nanoword.
.X..	All T commands apply to T-vector 2 of the current nanoword.
..X.	All T commands apply to T-vector 3 of the current nanoword.
...X	All T commands apply to T-vector 4 of the current nanoword.

Any statements not containing a position declaration will take on the same position attributes as the most recently encountered position declaration. A label statement positions the new nanoword in its K-Vector.

Position declarations may also specify that the T-Vector be stretched. In the list above, the appearence of an X indicates that the STRETCH nanoprimitive is not specified. If the "X" is replaced with an "S", the STRETCH nanoprimitive is specified to be active in the declared T-vector.

For the prototype system only, the letter "X" may be replaced with the letter "P" specifying an automatic hardware stretch is being activated by another nanoprimitive function, and therefore an explicit stretch is required only on production machines.

.S..	All T commands apply to T-Vector 2, and the STRETCH nanoprimitive is selected.
...P	All T commands apply to T-Vector 4, the STRETCH nanoprimitive will not be selected on the prototype QM-1.

6.3.2.2 NANOPRIMITIVE COMMANDS

The Nano-Assembler provides commands for the specification of all defined nanoprimitive fields in QM-1. These are divided into two classes, K commands and T commands, to correspond to the K-Vector and T-Vector portions of a nanoword. In general, each nanoprimitive command specifies the value of one, or more, fields in a K or T-Vector. K-Vector fields and T-Vector fields are summarized in sections 5.3.6 and 5.3.7, respectively, of this manual.

In the Version 1 Nano-Assembler, a nanoprimitive command may appear in one of four possible formats. These are:

```
NANOPRIMITIVE NAME
NANOPRIMITIVE NAME=<PRIMARY FIELD EXPRESSION>
NANOPRIMITIVE NAME(<SECONDARY FIELD EXPRESSION>)
NANOPRIMITIVE NAME(<SECONDARY EXPRESSION>)=<PRIMARY EXPRESSION>
```

Expressions are supported in the Version 1 assembler by a simple left to right, arithmetic evaluation. There is no defined operator precedence. Care must be taken in the coding of expressions, since the Version 2 assemblers will introduce basic multiplication and division precedence. All results are in two's complement, signed format. Version 1 does not check for field value overflow.

In many cases nanoprimitive functions require at least one associated nanoprimitive specification, in order to completely describe the action to be performed. For this reason provision is made for the primary nanoprimitive command name to specify both its own field value and, where necessary, a secondary field value. An example of this is:

```
READ CS ( CS ADDRESS SELECT )
```

Where "READ CS" sets the value 1 into the READ CS field, and the value of the expression in the secondary field (parenthesized) is placed into the corresponding CS ADDRESS selection field.

The Version 1 Nano-Assembler maintains a set of predefined variable symbol names, and values, within its symbol table (further discussed in section 6.3.2.3). These predefined symbols are provided for use within nanoprimitive expressions, and provide the corresponding values for most frequently used symbolic names (such as FCOD, G2, B, MPC, ADD, CARRY, etc.). The following tables describe the supported nanoprimitive commands, identify their secondary fields, and list those predefined symbolic names and values specifically provided for reference use in those fields.

The K-Vector nanoprimitive commands may appear following any K-Vector, or T-Vector, position declaration.

COMMAND NAME	FUNCTIONS
(Operating state control fields)	
LEGAL MICRO OP	- Sets the LEGAL MICRO OP ENTRY bit.
ALLOW NANO INTERRUPT	- Sets the ALLOW NANO INTERRUPT bit.
ALLOW MICRO INTERRUPT	- Sets the ALLOW MICRO INTERRUPT bit.
ALLOW INTS	- Sets both Allow Interrupt bits.
DIRECT MS ACCESS	- Sets the Direct Main Store Access bit.
HOLD	- Sets the HOLD bit (KALC, KSHC, KSHA, KS).
HOLD 2	- Sets the HOLD 2 bit (KA, KB).
SH STATUS ENABLE	- Sets the Shifter status enable bit.
ALU STATUS ENABLE	- Sets the ALU status enable bit.
SUPERVISOR	- Sets the Supervisory Instruction bit.
GENERATE INTERRUPT	- Sets the Generate / Clear Interrupt activation bit.
(Nano-Branch control fields)	
KN=<Expression>	- Sets the KN field to the value of <Expression>.
BRANCH(<Expression>)	- Sets the NANOBRANCH bit. Also allows the optional secondary field specification of the KN field to the value of <Expression>.
ALT BRANCH(<Expression>)	- Sets the ALTERNATE BRANCH Condition bit. Also allows KN specification as in BRANCH above.
PREP BRANCH	- Sets both the NANOBRANCH bit (BRANCH) and the ALTERNATE BRANCH Condition bit (ALT BRANCH).

(6 Bit data and function control fields)

The following 8 fields may each refer to any of the predefined symbols which are specifically oriented toward only one or two of those fields. This permits the placement of values, destined for a control field, into temporary holding fields for dynamic transfer during program execution.

KA=<Expression>	- Sets KA to the value of <Expression>.
KB=<Expression>	- Sets KB to the value of <Expression>.
KALC=<Expression>	- Sets KALC to the value of <Expression>.

ALU Control reference symbols provided are: ADD (11), SUB (6), DBL (14), INCR LEFT (17), DECR LEFT (0), PASS LEFT (37),

```
            PASS RIGHT (32), DECIMAL (40), AND (36), OR (33), XOR (31),
            NOT LEFT (20), NOT RIGHT (25), ZERO (34), ONES (23).
     KSHC=<Expression>        - Sets KSHC to the value of <Expression>.
           Shifter Control reference symbols provided are: LEFT (0), RIGHT (1),
            SINGLE (0), DOUBLE (2), CIRCULAR (0), LOGICAL (4), ARITHMETIC (10),
            RIGHT CTL (20), LEFT CTL (40).
     KS=<Expression>          - Sets KS to the value of <Expression>.
           References same as KT below.
     KT=<Expression>          - Sets KT to the value of <Expression>.
           Arithmetic / Shift condition (FIST) test mask reference symbols
            provided are: SLB (1), OVERFLOW (2), RESULT (4), SIGN (10),
            CARRY (20), SHB (40).
     KX=<Expression>          - Sets KX to the value of <Expression>.
           Machine state test condition reference symbols are: F ZERO (1),
            MS DATA (2), MS BUSY (4), PROGRAM CHECK (10), INDEX ZERO (20).
```

The T-Vector nanoprimitive commands may appear only after T-Vector position declarations.

```
     COMMAND NAME                FUNCTIONS
     ============             ================================================

     (Nanostore Control commands)

     READ NS                  - Sets READ NS bit.
     WRITE NS                 - Sets WRITE NS bit.

       The following 3 commands may all reference the same test conditions.

     TEST=<Expression>        - Sets the Test Specifier field to the value of
                                <Expression>.
           Test condition reference symbols are: S (2), NOT S (3), T (4),
            NOT T (5), X (6), NOT X (7).
     SKIP(<Expression>)       - Sets the GATE NS / SKIP "action" bit to zero.
                                If the secondary parameter is provided, sets
                                the Test Specifier field to the value of
                                <Expression>.  If no parameter is specified
                                (ie. SKIP,) a default value of 1 is used
                                (unconditional SKIP).
```

```
GATE NS(<Expression>)      - Sets the GATE NS / SKIP "action" bit to 1.
                             The secondary parameter is processed as for the
                             SKIP nanoprimitive, above.
GATE NS UNCONDITIONAL      - Sets the GATE NS UNCONDITIONAL bit (overrides
                             the conditional GATE NS function).

STRETCH                    - Sets stretch bit of the T-Vector.  May be used
                             in place of (or along with) the S type position
                             declaration.
LOAD NPC(<Expression>)     - Sets the NPC (Nano Program Counter) control
                             field to the value of <Expression>.  If the
                             secondary field is not specified the value 1
                             (CS) is used.
      NPC control reference symbols are: CS (1), KN (2), SEQ (3).

(Control Store access commands)

  The following 3 commands may all reference the same CS addressing names.

READ CS(<Expression>)      - Sets the READ CS bit.  The secondary field
                             specifies the CS ADDRESS selection code.  If
                             not specified the value 0 is used (CIA).
      CS ADDRESS selection reference symbols are: CIA (0), COD (1),
       MPC (2), INDEX (7).  Selection of MPC relative addresses is
       accomplished through arithmetic expression by: MPC+1 (3),
       MPC+2 (4), MPC+3 "B" (5), and MPC+4 "AB" (6).
WRITE CS(<Expression>)     - Sets the WRITE CS bit.  The secondary field is
                             the same as in READ CS, above.
CS ADDRESS=<Expression>    - Sets the CS Address Selection field to the
                             value of <Expression>.  See READ CS, above.
GATE CS                    - Sets the GATE CS bit.
INC MPC(<Expression>)      - Sets the INC MPC bit.  Sets the G SPEC field
                             to the value of <Expression>.  If the secondary
                             field is not specified a default value 14
                             (MPC PLUS 1) is used.
      There are no predefined INC MPC reference names at this time.  The
      following 4 commands set both the INC MPC bit and the proper value
      into the G SPEC field for the desired function.
MPC PLUS 1                 - Sets INC MPC and G SPEC = 14.
MPC PLUS 2                 - Sets INC MPC and G SPEC = 15.
MPC PLUS B                 - Sets INC MPC and G SPEC = 16.
MPC PLUS AB                - Sets INC MPC and G SPEC = 17.
```

LOAD R31	- Sets the LOAD R31 bit.

(Main Store access commands)

MSGO	- Sets the MSGO bit.
FETCH MS	- Sets the MSGO bit.
MSRS	- Sets the MSRS bit.
WRITE MS	- Sets the MSRS bit.
READ MS	- Sets both MSGO and MSRS bits.
RMI=<Expression>	- Sets the RMI field to the value of <Expression>.
GATE MS(<Expression>)	- Sets the GATE MS bit. If the secondary field is specified, sets the RMI field to the value of <Expression>.

(18 Bit data control commands)

LOAD ES	- Sets the LOAD ES bit.
GATE ES	- Sets the GATE ES bit.
GATE SH	- Sets the GATE SH bit.
GATE ALU	- Sets the GATE ALU bit.
INDEX(<Expression>)	- Sets the INDEX ALU gate bit. If the secondary field is specified, the value of <Expression> is placed into the FSEL2 field (for INDEX ALU function selection).

INDEX ALU function reference symbols are: SUB (6), ADD (11).

INDEX REG(<Expression 2>)=<Expression 1>	- Sets the value of the primary field <Expression 1> into the AUX2 field (Local store register selection) and the value of the secondary field <Expression 2> into the AUX3 (Index register selection). Either one or both parameters may be specified.
CARRY CTL=<Expression>	- Sets the Carry Control field to the value of <Expression>. Specification of the desired carry control functions may be accomplished with any of the 7 following commands.
CLEAR CIH	- Sets the CARRY CTL field to the value 1.
SET CIH	- " " " " " " " " 2.
ALU TO BOTH CIH AND COH	- " " " " " " " " 3.
ALU TO COH	- " " " " " " " " 4.

SET COH	- " " " " " " " " 5.
CLEAR COH	- " " " " " " " " 6.
SH TO COH	- " " " " " " " " 7.

(6 Bit data control commands)

INO	- Sets the INO bit (AUX to F register).
OUTO	- Sets the OUTO bit (F register to AUX).
AUXO(<Expression 2>)=<Expression 1>	
	- Sets the AUXO field to the value of <Expression 1>. If the secondary field is specified the INO field is set to the value of <Expression 2>. The INO field may contain only a 0 or 1.
FSELO(<Expression 2>)=<Expression 1>	
	- Sets the FSELO field to the value of <Expression 1>. If the secondary field is specified the OUTO field is set to the value of <Expression 2>. The OUTO field may contain only a 0 or 1.
IN1	- Sets the IN1 bit.
OUT1	- Sets the OUT1 bit.
AUX1(<Expression 2>)=<Expression 1>	
	- Sets the AUX1 and IN1 fields as in AUXO above.
FSEL1(<Expression 2>)=<Expression 1>	
	- Sets the FSEL1 and OUT1 fields as in FSELO above.
F(<Expression>)	- Sets the FSEL1 field to the value of <Expression>. This command format is provided for specification of the F register to be tested under the conditional GATE NS / SKIP on "F ZERO" (KX test condition).
IN2	- Sets the IN2 bit.
OUT2	- Sets the OUT2 bit.
AUX2(<Expression 2>)=<Expression 1>	
	- Sets the AUX2 and IN2 fields as in AUXO above.
FSEL2(<Expression 2>)=<Expression 1>	
	- Sets the FSEL2 and OUT2 fields as in FSELO above.
AUX3=<Expression>	- Sets the AUX3 field to the value of

<Expression>.

Several of the above 6 bit data control fields are used in support of other functions, such as INDEX ALU, and may also be set as primary or secondary fields of other nanoprimitive commands.

A set of predefined reference symbols is provided for use by all F select fields. These symbols are:

```
 FMIX  (0), FMOD  (1), FCIA  (2), FAIL  (3), FCID  (4), FAIR  (5),
 FCOD  (6), FADD  (7), FSID (10), FSOD (11), FEID (12), FEOD (13),
 FEIA (14), FEOA (15), FINV (16), FACT (16), FLIV (17), FMPC (20),
 FIDX (21), FIST (22), FIPH (23), G0   (24), G1   (25), G2   (26),
 G3   (27), G4   (30), G5   (31), G6   (32), G7   (33), G8   (34),
 G9   (35), G10  (36), G11  (37).
```

A set of predefined reference symbols is provided for specific source and destination AUX-select fields. These symbols are: A (0), B (1), C(2) for AUX1 source and all destinations, KX (2) for AUX2 source, KA (3), KB (4) for AUX0 and AUX2 source and all destinations, KT (4) for AUX1 source, GSPEC (5) for all source, ALUF (6) for AUX0 source, IO ID (7) for AUX0 source, INCF (6) for AUX1 and AUX2 source, DECF (7) for AUX1 and AUX2 source, KSHC (5) for AUX0 and AUX3 destination, KALC (6) for AUX0 and AUX1 destination, KSHA (7) for AUX1 and AUX3 destination, KS. (7) for AUX0 destination, KX. (5) for AUX1 destination, KT. (6) for AUX3 destination.

These explicit 6 bit data transfer controls may be used as follows:

1. Transfer B field (of R31) to FSID.

```
X...  AUX0(1)=B, FSEL0=FSID, .............
```

2. Transfer KX to KALC (ALU control) through FIPH (phantom F).

```
.S..  AUX2(1)=KX, FSEL2(1)=FIPH, AUX3=KALC, ...............
```

More concise command declarations may be obtained using the implicit 6 bit data transfer commands, as described in section 6.3.2.4.

(General Control and Input / Output Commands)

G(<Expression>)	- Sets the G SPEC field to the value of <Expression>. This format is provided for support of 6 bit data transfer commands (see section 6.3.2.4).

Special G SPEC selection reference symbols are: G KSHA (14), G B (15), G KS (16), G KX (17).

G SPEC=<Expression>	- Same as G(<Expression>) above.
XIO(<Expression>)	- Sets the XIO bit. If the secondary field is specified, sets the value of <Expression> in G SPEC.
RIO(<Expression>)	- Sets the RIO bit. If the secondary field is specified, sets the value of <Expression> in G SPEC.
AUX ACTION	- Sets the AUX ACTION bit. (Activates external commands via use of F register "FACT").

6.3.2.3 VARIABLE SYMBOLS AND CONSTANTS

A variable symbol table is provided, in the Nano-Assembler, to permit user definition of their own symbolic representations for most reference symbols (see section 6.3.2.2 for lists of reference symbols). In addition many parameters may be specified using user selected variable names in place of constants, permitting easy modification of source programs. A user variable may be redefined at any point in the assembly. Specification is accomplished through arithmetic assignment statements.

The variable symbol to be defined (or redefined) appears as a symbolic name, to the left of an equal (=) sign. This name must not be the same as any currently defined nanoprimitive command name. The value to be placed into the variable is computed from a simple arithmetic expression, to the right of the equal sign. The Version 1 Nano-Assembler supports four arithmetic operators. All multiplication (*) and division (/) operations are performed before any additions (+) or subtractions (-). [Version 1, Level 2, recognizes no operator precedence.] Expressions may consist of any other variable names and constants.

Constant values may be specified in octal or decimal notation. A decimal constant consists of a string of digits followed by a decimal point (.). An octal constant consists of a string of octal digits (0 through 7) followed by an operator or delimiter. Trailing blanks are ignored.

```
     Decimal constants:  1.,  20.,   -30790.,  0.
     Octal constants:    1,   20,   37707,   -77077,  0
```

All negative values are represented in two's complement notation. Care must be taken in setting up mask values, since -0770 is not the true complement of the value 7007 (-0770 appears as 777010 in 18 bit signed notation). The Version 1 assembler supports numeric values between +32,767 and -32,768.

Variable assignment expressions may be placed within their own fields on any type of statement, excluding comment statements, or on their own statements. The following are some examples of variable symbol use.

1. Variable name for use in CS ADDRESS selection:

```
     S...  B FIELD = 3, READ CS (MPC + B FIELD), GATE CS
```

2. Setting up alternate names for special purpose F or G register reference:

```
     F.ZERO = G0  "SOURCE OF CONSTANT 6 BIT ZERO AS AN F REGISTER SELECT"
     G.ZERO = G0-20  "SOURCE OF CONSTANT ZERO FOR G SPEC SELECTION"
     ..S.  G(G.ZERO), AUX0(1)=GSPEC, FSEL0=FIST, "CLEAR FIST"
           FSEL1(1)=F.ZERO, AUX1=A, "CLEAR A OF R31"
```

3. Selecting address references:

```
           NEXT.INSTR = N. + 1  "N. == CURRENT NANO-LOCATION-COUNTER"
     ....  BRANCH (NEXT.INSTR)
             "or optionally"
     ....  BRANCH (N. + 1)  "SETS KN = N. + 1"
```

6.3.2.4 6 BIT DATA TRANSFER COMMANDS

Specification of 6 bit data transfers between AUX fields and F registers, and direct F register modification, requires the use of a minimum of 3 T-Vector nanoprimitive fields. In addition, only the A, B, and KA fields are uniformly accessible as sources and destinations in all 3-6 bit control groups.

The nanoprogrammer has the responsibility of knowing how many occurrences of each AUX exist in a T-Vector, but the assembler can determine which control group to use for each 6 bit data operation. This automatic selection is in effect when using the 6 bit data transfer commands. There are four formats available for stating these commands, as follows.

1. (SOURCE AUX NAME)->(Expression)
2. (SOURCE AUX NAME)->(Expression)->(DESTINATION AUX NAME)
3. (SOURCE AUX NAME)->(DESTINATION AUX NAME)
4. (Expression)->(DESTINATION AUX NAME)

[Version 1, Level 2, is not compatible with the above definition. Transfer command declarations must be stated, by the nanoprogrammer, in the best fit, left to right, order, for correct placement into control groups.]

[Version 1, Level 2, (Expression) may not be used, as only a single variable symbol name will be recognized in these fields. See example below.]

(Expression) is equivalent to all legal arithmetic expressions allowed in the explicit F-select field commands, as described in section 6.3.2.2. Both the (SOURCE AUX NAME) and (DESTINATION AUX NAME) components must be one of those listed in the tables below. Format I specifies an AUX field to F register transfer, or direct F register modification (ie. increment F). Format 4 specifies an F register to AUX field transfer. Format 2 specifies a replacement operation, where the AUX transfer parallels the F transfer, usually exchanging the two fields. It may also indicate a pass operation, when the value of (Expression) is FIPH, where the source AUX field is transfered directly to the destination AUX field. Finally, format 3 is equivalent to format 2, specifically invoking the use of FIPH to pass one AUX field to another.

The following table lists all source AUX names and their actual control group occurrences:

SOURCE AUX NAME	VALUE	GROUP NUMBERS		
A	0	0	1	2
B	1	0	1	2
C	2		1	
KA	3	0	1	2
KB	4	0		2
KX	2			2
KT	4		1	
INCF	6		1	2
DECF	7		1	2
IO ID	7	0		
ALUF	6	0		
G	5	0	1	2

The following table lists all destination AUX names and their actual control group occurrences:

DESTINATION AUX NAME	VALUE	GROUP NUMBERS		
A	0	0	1	2
B	1	0	1	2
C	2	0	1	2
KA	3	0	1	2
KB	4	0	1	2
KS	7	0		
KX	5		1	
KT	6			2
KSHC	5	0		2
KALC	6	0	1	
KSHA	7		1	2

Examples of transfer commands:

1. Transfer the A field of R31 to FMOD.
```
A->FMOD  "Equivalent to: AUX0(1)=A, FSEL0=FMOD"
```
2. Exchange the contents of the B field with FIST.

```
    B->FIST->B  "Equivalent to:  AUXO(1)=B, FSELO(1)=FIST"
3.  Transfer KT to G2, while transfering G2 to KSHA.
    KT->G2->KSHA  "Equivalent to: AUX1(1)=KT, FSEL1=G2, "
                  "                FSEL2(1)=G2, AUX3=KSHA"
4.  Transfer KX directly to KALC.
    KX->KALC  "Same as KX->FIPH->KALC"
5.  Transfer an F register one greater than F.WORK to KX.
    F.WORK+1->KX  "Equivalent to: FSELO(1)=F.WORK+1, AUXO=KX."
```

[Version 1, Level 2 restriction, (Expressions) may not be used in 6 bit transfer commands. Only single symbolic names may be referenced. In order to accomplish that shown in example 5, a temporary name must be used to hold the F register value. (ie. TEMP=F.WORK+1, TEMP->KX).]

6.3.3 CONTROL STATEMENTS

Control statements consist of one or more control fields. Though all control field operators will be recognized on other statement types, the following caution should be observed. All control fields terminate the nanoword currently being assembled. In addition, statements containing some of the control fields will not be listed. Two classes of control statements are defined for Version 1: Assembly control, and listing control.

6.3.3.1 ASSEMBLY CONTROL STATEMENTS

Only one assembly control statement is defined for Version 1. The "END" control operator indicates the end of the last source statement of the current assembly. It should be placed by itself on a source statement. It will always be listed, regardless of listing controls specified.

6.3.3.2 LISTING CONTROL STATEMENTS

There are 4 listing control statements defined for Version 1. Each terminates
e h l on
£

LIST OFF	No parameters. Remainder of statement is processed. The statement containing "LIST OFF" will not be listed, as well as all following statements until one with a "LIST ON" control command is encountered. Lines in error will be listed unconditionally.
LIST ON	No parameters. Remainder of statement is processed. Reverses the effect of a previous "LIST OFF" command.
"."	(Single period) No parameters. Remainder of statement is processed. "." simply indicates the end of the current nanoword. It is used optionally to trigger normal assembly generated end of word listing information. This permits insertion of extra lines and comments ahead of the next word definition. When not terminated by command, the label statement of the next nanoword triggers the generated listing information.
EJECT	No parameters. Remainder of statement is ignored. "EJECT" is never listed but will cause an eject to top of next page, while "LIST ON" is in effect.

6.4 OPERATION [VERSION 1, LEVEL 2, UNDER NCS ONLY]

6.4.1 INVOCATION

The Nano-Assembler is initiated by command at the system console. Entering the name "NA," will begin assembler execution. The request for input file name will be displayed as "INPT=". Respond with the disk data file name, followed by a comma. The request for binary output file name will be displayed next as "BIN=". Respond with either the file name to be used, followed by a period, or just a period to indicate that no output file is desired. The escape key (ESC) may be used to cancel a partially entered file name, in order to correct keying mistakes.

If any errors are detected during the assembly the message "ASSEM. ERRORS" will be displayed at the end of processing.

6.4.2 ERROR FLAGS

Errors are indicated through use of single character codes placed on the line immediately following the line in error. Each code will appear directly under the symbol or character in error. There are three classes of error detection: LEXICAL, SYNTACTIC, and GENERAL.

Lexical errors are detected during initial scan over the source statements. Any illegal characters or unrecognized character sequences are flagged with a digit as follows:

CODE	DESCRIPTION	
0	Illegal first character in a field.	
1	Illegal character within a symbolic name.	
2	Illegal character within a numeric string.	
3	Illegal octal number.	
4	Internal error in lexical analyzer.	***
5	Expected operator or field delimiter missing.	
6	Syntax table full (Expression overly complex)	***

*** Report these errors to: The Systems Software Division, NANODATA CORPORATION, 2457 Wehrle Drive, Williamsville, New York 14221. Please provide as much supporting material as possible (within reason), listings, decks, dumps, etc.

Lexical errors cause a skip to the next comma, or end of statement, whichever occurs first. This skip is indicated by a string of hyphens from the error code to the end of the skip, on the error flag line.

Syntactic errors are detected while the assembler is attempting to classify each field (nanoprimitive, label, arithmetic expression, control, etc.). Each error receives a letter code as follows:

CODE	DESCRIPTION
A	Improper first element in a field.
B	First element improperly terminated.
C	Improper first element in a nanoprimitive secondary field.
D	Improper first element in an arithmetic expression.
E	Illegal element, or improper termination, in a nanoprimitive secondary field.
F	Improper element in secondary field.
G	Secondary field not followed by legal delimiter.
H	Improper element within an arithmetic expression.
I	Improper arithmetic operator within an expression.
J	Illegal component in a 6 bit data transfer command.
K	" " " " " " " " "
L	" " " " " " " " "
M	Improperly formatted, three element, 6 bit data transfer command.

General error flags are set by various statement class, and field type, processes. These are listed below:

CODE	DESCRIPTION
?	Unrecognized nanoprimitive command, control operator, or pseudo command operator.
=	Warning; that a redefinition of one of the predefined reference variables has occurred.
$	Internal assembler control error. *** (above)
M	Multiply defined symbolic label name, or multiply declared nanoprimitive field.
T	T-Vector nanoprimitive declared within K-Vector statement range.

W	Nanoprimitive declared outside the range of a nanoword.
P	6 bit data transfer command will not fit in the current T-Vector. All acceptable groups in use.
X	Undefined variable name in arithmetic expression.
U	Unnecessary primary or secondary nanoprimitive command specification.
I	Incomplete nanoprimitive specification. A required primary or secondary field declaration is missing.

General; listable error messages:

*** BINARY OUTPUT FILE FULL ***	- Specify a larger output file.
* ERROR *	- Left side of all error flag lines.

Console error messages:

BAD NAME	- Illegal disk file name entered.
FILE NOT FOUND	- Try another name, disk or volume.
MALFUNCTION	- Usually are hardware failure during assembly.
SYMBOL TABLE OVERFLOW	- Assembly requires more memory space.
ASSEM. ERRORS	- One or more errors during assembly,

7 NANOPROGRAMMING EXAMPLES

7.1 BASIC MODEL NANOPROGRAMS

One of the most basic nanoprograms that can be actually used is illustrated below. This nanoprogram implements a WAIT microinstruction that cycles until an interrupt occurs.

```
WAIT:
....  LEGAL MICRO OP ENTRY, ALLOW NANO INTERRUPT, ALLOW MICRO INTERRUPT
X...
.X..  READ NS
..X.  GATE NS
...X
```

The K-vector commands specify that this is a legal microinstruction and that all interrupts are allowed following execution of this nanoword.

In the second T-step, nanostore is read using the address that remains in the NanoProgram Counter (NPC) following the beginning of execution of the nanoword. In the third T-step, the nanostore word read is gated into the control matrix. It will begin execution following the third T-step (the last T-step is unused.)

Since the successive nanostore words read are the same as the first one triggered by execution of the WAIT microinstruction, this nanoprogram cycles repeatedly until an interrupt occurs.

When an interrupt occurs (and is accepted by the machine), the above sequence is suspended. The next nanoword read and placed into execution is the one addressed by the particular interrupt that has occurred.

The first T-step is not used for the READ NS nanoprimitive, in the example, since interrupts are allowed following the execution of this nanoword. Interrupt address selection requires at least one T-period, and cannot begin until at least one of the allow interrupt K-Vector nanoprimitives is recognized. The earliest interrupt allow recognition time is at the leading edge of T1. Since READ NS address selection is also a leading edge function it may not be executed in T1, unless it is selecting a nanobranch (KN) address.

In the example above, no access is made to control store since the same nanoword is repeatedly read. Consequently, nothing has to be done to change the microprogram counter. Its current value still indicates the control store WAIT instruction that initiated the nanoprogram. In the more typical situation, successive control store instructions would be read and used to address different nanoprograms that implement the successive instructions. The next example shows the implementation of a No-OPeration microinstruction that includes reading the next microinstruction from control store and updating the microprogram counter.

```
NOP:
....  LEGAL MICRO OP ENTRY, ALLOW NANO INTERRUPT, ALLOW MICRO INTERRUPT
X...  READ CS(MPC+1)
.X..  LOAD NPC(CS), MPC PLUS 1  "INC MPC +1; THIS IS A COMMENT"
..X.  READ NS
...X  GATE NS, LOAD R31
```

In this nanoprogram, the convention has been established that at nanoprogram entry, the MPC (one of 4 available local store registers) points to the currently executing microinstruction in control store. The actual MPC in use is determined by the contents of FMPC when the nanoprogram is executed. It is further defined that the NOP microinstruction is one word long. (This is a reasonable assumption since no parameters are needed in a NOP!) Thus during T1, control store is read at the address one greater than the current contents of the MicroProgram Counter. Thus the next microinstruction in sequence is being read.

During T2, the MicroProgram Counter is updated. This assures that it will point to the next microinstruction when that microinstruction begins execution. Simultaneously, this new microinstruction from control store (along with the contents of the page register in FIDX) is used to load the NanoProgram Counter (NPC).

From this point on, the nanoprogram is similar to the first example. The nanostore word that begins the implementation of the next microinstruction is read and gated into the control matrix. Simultaneous with this last action, the A and B parameters of the new microinstruction are loaded into R31. The new nanoprogram then begins execution.

This example is shown as a model since it consists of a basic set of nano-primitives that will be common to many nanoprograms. It forms a basis for all one word nanoprograms that implement one-word microinstructions under the nanoprogramming conventions mentioned. The next example will illustrate another of the many possible sets of conventions that may be selected.

One possible nanoprogramming convention which can be established is that each nanoprogram would expect the next sequential word from control store to be available at the beginning of its execution (on the COD bus) and that each nanoprogram would be responsible for maintaining this prefetch convention for the next nanoprogram. The following example shows a possible form of the NOP when programmed according to this new convention.

```
NOPLA:
....  LEGAL MICRO OP ENTRY, ALLOW INTS
X...  LOAD NPC(CS)
.X..  READ NS, READ CS(MPC+2), INC MPC
..X.  GATE NS, LOAD R31
```

Since the next word from control store is already available, we can immediately load the NanoProgram Counter in T1. At the same time, we start the operations necessary to set up a similar situation at the end of the nanoprogram by reading control store. The address used is MPC+2 since we need to read the word ahead by two from the currently executing microinstruction.

The remainder of the example proceeds as before. At the end of T3, we are ready to execute the next nanoprogram and the next sequential word from control store is available on the COD bus. As before, the MicroProgram Counter points to the currently executing microinstruction.

This example has been given for illustrative purposes only; the advantages of a Control Store prefetch are not explored here. It does illustrate, however, some of the possible freedom available to the nanoprogrammer in selecting the conventions that best suit his purposes.

Now an example will be given showing a very simple nanoprogram which actually uses the microinstruction parameters. This is an implementation of the MOV A,B microinstruction that causes the contents of local store register B to be moved to local store register A, where A and B are the parameters in the MOV microinstruction.

```
MOV:
....  LEGAL MICRO OP, ALLOW INTS
....  KSHA=0, KSHC=0
X...  READ CS(MPC+1), A->FSOD, B->FSID
.X..  LOAD NPC(CS), MPC PLUS 1
..X.  READ NS, GATE SH
...X  GATE NS, LOAD R31
```

This nanoprogram uses the shifter as a path to accomplish the move. Thus the main part of the program is the setting of the shifter bus controls FSID and FSOD, followed by the gating of the shifter to actually cause the transfer. Note that the controls are set up in T1 but the gate operation is deferred until T3. This is necessary in order for the shifter output to be stable.

Since the shifter is used merely as a path to accomplish the move, no shifting is required. The shift amount is explicitly set to zero with KSHA=0. This is done to make the example clear; it is unnecessary, as is KSHC=0, since the default value for these fields is zero when they are not mentioned.

Note that all of the nanoprimitives of the NOP nanoprogram are included in the example MOV nanoprogram. This is done since some scheme for fetching and sequencing of microinstructions is necessary. The scheme illustrated in the NOP nanoprogram is one such scheme and is satisfactory for the instruction presented. The prefetch scheme would work as well as is shown below.

```
MOVLA:
....  LEGAL MICRO, ALLOW INTS
....  KSHA=0, KSHC=0
X...  LOAD NPC(CS), A->FSOD, B->FSID
.X..  READ NS, READ CS (MPC+2), MPC PLUS 1
..X.  GATE NS, GATE SH, LOAD R31
```

In fact, this also illustrates a possible advantage of the prefetch since the nanoprogram is shorter by one period.

At this point, it is useful to introduce a shorthand notation to simplify writing sets of nanoprograms. By predefining a set of nanoprimitives that appear frequently, they may be invoked by name when needed. This is illustrated in the next section.

7.2 USE OF PREDEFINED NANOWORDS

[NOTE: In current versions of the Nano-Assembler no support is provided]
[for fetching predefined nanowords. It is likely that this support will be]
[added in later versions. Until then, predefined nanowords are described in]
[this document strictly for illustrative purposes.]

It is useful to have a way to condense the description of sets of nanoprograms having frequent repetition of the same nanoprimitive sequences. This is done by reference to predefined nanowords. Any nanoword previously defined and labeled may be invoked to cause all of the predefined bits to be set in the nanoword in which the label of the predefined nanoword is mentioned.

For example, the MOV nanoprograms of the previous section may be written:

```
MOV:
....  NOP, KSHA=0, KSHC=0
X...  A->FSOD, B->FSID
..X.  GATE SH
```

Here the fetch sequence is invoked by reference to NOP, provided the NOP nanoword has been defined as shown in the previous section. All of the bits on in that nanoword will be set on in this nanoword.

Similarly, the prefetch sequence can be invoked instead by changing NOP to NOPLA.

```
MOVLA:
....  NOPLA, KSHA=0, KSHC=0
X...  A->FSOD, B->FSID
..X.  GATE SH
```

This notation will be used extensively in the examples that follow. It makes the examples shorter and allows one to concentrate on the parts of each nanoprogram that are novel to each new example. Considerable care must be exerted, however, to avoid combinations that are not consistent.

The simple fetch procedure, described in the beginning of this chapter, will be used in most of the examples. It is formally coded as follows:

```
FETCH:
....  LEGAL MICRO OP ENTRY, ALLOW NANO INTERRUPT, ALLOW MICRO INTERRUPT
X...  READ CS (MPC+1)
.X..  LOAD NPC (CS)
..X.  READ NS, MPC PLUS 1
...X  GATE NS, LOAD R31
```

The INC MPC nanoprimitive (MPC PLUS 1) has been placed in T3 so that the previous contents of MPC may be used during T1, T2 or T3 in nanoprograms that use FETCH.

To illustrate the use of this predefined nanoword, the following nanoprogram implements a LD A,B microinstruction that loads from control store, into local store register A, the word addressed by the contents of local store register B.

```
LD:
....  FETCH
X...  A->FCOD, B->FCIA
..X.  READ CS (CIA)
...X  GATE CS
```

In this example, the essential but previously discussed parts of the nanoprogram are condensed into the reference to FETCH. Then, the parts of the nanoprogram important to make the LD work can more clearly be seen.

7.3 CONTROL STORE ACCESS NANOPROGRAMS

The previous example shows how one may implement an instruction to load a word from control store into a local store register, using an address already in a local store register. The examples in this section will illustrate other ways in which control store may be addressed and accessed.

The address of the desired control store data location may itself be fetched from control store. Two simple access techniques can be implemented. First of all, one could treat the lower 64 (decimal) locations of control store as a set of directly accessed special registers. Second, the data address could be found at some control store location relative to the load instruction itself. The most accessible location, in this case, would be the next sequential control store word.

The following example illustrates a microinstruction that reads the control store location identified by the value of B, into local store register A.

```
LDCSR:
....  FETCH, KA=31.
X...  A->FCOD, KA->FCIA, FIPH->A
.X..
..X.  READ CS (CIA)
...X  GATE CS
```

The Load-Via-Control-Store-Register instruction uses the predefined FETCH procedure to accomplish its next microinstruction access. Care must be taken not to interfere with that logic, especially since both FETCH and the instruction execution logic access control store.

T1 of our example sets up the F-registers for the data access. Local store register A is connected to the COD bus, register 31 (decimal) is connected to the CIA bus, and the original content of the A field is zeroed leaving only the B field remaining in R31. FIPH is a source of zeroes when no source AUX is transmitted to it, within the same T-step. Note that the C field of R31 does not need to be cleared, as the conventional microinstruction initiation sequence clears C while loading A and B (LOAD R31 nanoprimitive).

T1 and T2 are used by the FETCH sequence to access control store. We may use control store for our data access beginning in T3. Local store register 31 is now used for addressing control store. Note also that we are accessing both nanostore and control store simultaneously during T3 and T4. The FETCH sequence is reading the next nanoprogram while the LDCSR logic is reading the data word.

To be able to address any control store location the Load-Via-Next-Word instruction may be defined. In this case the microinstruction may be considered as being two words in length, where the second word contains the absolute, 18 bit control store address. The data is loaded into the local store register identified by the A field of the first word of the instruction.

There are many ways of organizing this nanoprogram. In order to use the FETCH predefinition we must define an additional programming convention. If we enforce the rule that any nanoprogram that uses FCOD must restore it to the value 31 (decimal) prior to completion, we may then execute a GATE CS nanoprimitive in T1 with the knowledge that the COD bus is already connected to R31.

```
LDNW:   "WITH FCOD=31 CONVENTION"
....  FETCH, KA=31.
S...  GATE CS, A->FCOD, KA->FCIA
.S..  READ CS (MPC+2), MPC PLUS 1
..X.  READ CS (CIA)
...X  GATE CS, KA->FCOD
```

The FETCH sequence normally initiates the read of CS location MPC+1 in T1, then loads NPC with the results of the read in T2. By stretching T1 the READ CS (MPC+1) is completed within T1, and our GATE CS will place the content of MPC+1 into local store register 31 (by our FCOD convention). Thus, by the end of T1 our data address resides in R31. Since we have changed R31 by the end of T1 we must extract all necessary A and B field information during T1. B is not used, but A must be transfered to FCOD (we have now altered FCOD).

The FETCH sequence expects the next instruction to be available on the COD bus by the end of T2. We can provide this by stretching T2, and executing a READ CS(MPC+2) at the beginning of that T-step. T3 and T4 now can function the same as in the LDCSR instruction above. The only difference will be our restoration of FCOD to the value 31, as required by our new convention.

It is possible to define an LDNW instruction without the above convention, but the FETCH definition may not be applied. We also will require two additional T-periods as follows.

```
LDNW:    "WITH NO CONVENTIONS"
*       INSTRUCTION EXECUTION CODE "" INSTRUCTION FETCH CODE
....                               ""
S...    READ CS(MPC+1), A->FCOD    ""
.S..    READ CS(COD), GATE CS      ""
..S.                               "" READ CS(MPC+2), MPC PLUS 2, LOAD NPC(CS)
...S                               "" READ NS, GATE NS, LOAD R31
```

In T1 above we read the word from MPC+1 onto the COD bus, but we do not gate it into local store. We also set up FCOD. Then, in T2, we point the control store address select directly at the data on the COD bus, and use this as our data address. T2 must complete the data access in order to free control store for a third read, which must begin in T3 if we are going to complete this nano-program within one nanoword.

Note that in both of the LDNW examples we perform three READ CS operations. One is required to read the data address, another to access the data, and the third to fetch the next instruction. Only the order of events has changed between the two methods. A LDNWLA (look ahead) instruction sequence could also be written in a similar fashion, saving at least one T-period. We leave the proof of this as an exercise for the reader.

The following example illustrates the writing of control store. The STCSR instruction is the counterpart of LDCSR, and will store the local store register identified by the A field, into the control store location identified by the absolute value of B.

```
STCSR:
....  FETCH, KA=31.
X...  A->FCID, KA->FCIA, FIPH->A
.X..
..X.  WRITE CS (CIA)
...X
```

That's all there is to that. FCID is set up in T1, and the WRITE CS is performed during T3 and T4. Everything else is the same as in the LDCSR instruction, above.

7.4 MICROINSTRUCTION BRANCH NANOPROGRAMS

There are several QM-1 hardware functions that simplify the definitions of branch, or jump, instructions. Only those basic instructions will be explored in this section. More advanced instruction formats (ie. branch and link) will be discussed in section 7.6. Three branching examples are shown below. Although all of the nanoprimitives of the FETCH sequence are used in one of the examples, the functional meanings of the use of each primitive differs extensively, therefore no predefinitions are used in that example.

The ability to increment the micro-program-counter by the values 1 or 2, and to add the two's complement values of either the 6 bit B field or 11 bit A and B field concatenation to the MPC, simplify the definitions of some forms of branch instructions. The first example executes a branch that is relative to the address of the branch microinstruction itself. BPREL (Branch Program-Counter Relative) allows a forward branch of up to 1,023 (decimal) locations, and a backward branch of up to 1,024 locations. The AB field concatenation hardware permits the following encoding.

```
BPREL:
....
X...  "SETTLING TIME FOR COMPLETION OF MPC+AB CIRCUIT COMPUTATION."
.S..  READ CS (MPC+AB "AB=4"), LOAD NPC (CS), MPC PLUS AB
..X.  READ NS
...X  GATE NS, LOAD R31
```

The A and B fields of R31 receive the new microinstruction parameters at the start of T1, therefore some time must be allowed for the new AB value to be applied to the MPC+AB addition circuit. Following T1 the proper value is available for use as a control store address and for gating into the MPC register. T2 is used to read the next microinstruction, and to route its address into MPC. NPC is also loaded with the nanostore address. T3 and T4 then read and gate the new nanoprogram, as in most previous examples.

Frequently the branch address will be found in a register. For example; entry into a subroutine may leave the return address in a register, requiring a simple transfer of control to the instruction at that address upon subroutine exit. A BR (Branch-Via-Register) instruction can be defined as follows.

```
BR:
....  KA=LS.MPC
X...  KA->FSOD, A->FSID, A->FCIA
.X..  "ALLOW CIA AND SHIFTER SETTLING TIME."
..S.  READ CS (CIA), LOAD NPC (CS), GATE SH
...S  READ NS, GATE NS, LOAD R31
```

In this example the MPC local store register has been symbolically named LS.MPC, and is defined in the KA field. Should the actual MPC be a variable (remember, there are 4 possible MPC registers) a convention may be specified whereby the current MPC register number could be found within a specific G-register. In that case it could be copied from the G holding register to FSOD as follows.

```
S...  G(G.MPC), G->FSOD, A->FSID, A->FCIA
```

The shifter is being used to pass the actual address from the local store register, defined by A, into the current MPC. Control store address selection requires that the actual address be available at least one T-period prior to the actual READ CS operation. The address will not be stable, in the MPC, soon enough to be referenced by a READ CS (MPC), especially in a one nanoword program. We do, though, have the address available in its original local store register, and may reference that register for CS addressing via the CIA bus. Therefore A->FCIA in T1 sets up the appropriate connection soon enough to allow our READ CS (CIA) in T3.

The complete control store access is done in a stretched T3, allowing NPC to be set up for use by the READ NS in T4. T4 is also stretched, allowing both initiation and completion of the nanostore access.

Another form of branching allows programmed decision making. The next example shows a conditional branch instruction. In this case no address is passed to the instruction, as it requires no operands. Our instruction will "skip" over the next sequential instruction if the result of the last arithmetic operation instruction was positive, or zero. The Skip-On-Plus instruction will test the SIGN indicator of F-register FIST. If the SIGN indicates a zero (positive) it will skip the next instruction. If SIGN is a one (negative) it will execute the next sequential instruction.

```
SKIP ON PLUS:
....  KS = SIGN
*    FIRST READ THE NEXT SEQUENTIAL INSTRUCTION.
S...  READ CS (MPC+1), LOAD NPC (CS), SKIP (NOT S)
*    SKIP OVER T2 IF (FIST .AND. KS) ARE NOT ZERO.
.S..  READ CS (MPC+2), LOAD NPC (CS), MPC PLUS 1
..S.  READ NS,                        MPC PLUS 1
...X  GATE NS, LOAD R31
```

T1, above, reads the next sequential instruction (MPC+1) and prepares the NPC in case T2 is skipped. If T2 is not skipped it will read the microinstruction at MPC+2, and execute an extra increment MPC operation. The LOAD NPC (CS) in T2 will replace the value set into the NPC in T1. The conditional skip tests the result of the logical "AND" operation between the KS field of the nano-program and FIST. The skip will take effect if the result of the "AND" is "NOT" zero.

At T3 the MPC will be incremented once, unconditionally. If T2 had not been skipped the MPC will be 2 greater than at the beginning of the Skip-On-Plus instruction. T3 and T4 complete the access to the appropriate nanoword, as selected during T1 and T2.

7.5 ARITHMETIC NANOPROGRAMS (SHIFTS)

This section will illustrate several examples of ALU, INDEX ALU, SHIFTER, and F register increment / decrement facilities. Although it is not shown in any one example, all of these functional units may be used simultaneously and independently. Only the standard ALU and SHIFTER may be used for combined functions, in support of double precision shifts and related ALU / shift operations.

The independence of the ALU and SHIFTER may be shown in the SWAP instruction. SWAP simultaneously exchanges the contents of the local store register identified by the A field with the register identified by the B field. The ALU is used to transfer R(B) to R(A) while the SHIFTER is used to transfer R(A) to R(B). The ALU operation command is PASS LEFT, which uses only the left ALU input and transmits the input data to the ADD bus without modification.

```
        SWAP  A,B      [ R(A)=R(B); R(B)=R(A) ]
SWAP:   "SWAP THE CONTENT OF THE A AND B LOCAL STORE REGISTERS"
....  FETCH, KALC = PASS LEFT
X...  A->FSID, B->FSOD, B->FAIL
.X..                    A->FAOD
..X.  GATE SH, GATE ALU
...X
```

No KSHC field declaration was needed as the default SHIFTER condition (0) is SINGLE, LEFT, CIRCULAR. A zero shift amount is also the KSHA field default, and will always result in an unmodified SHIFTER output. T1 is used to set up both input F registers, in order to begin the propagation of the data through the functional units as early as possible. Two T-periods are required before the data is stable on the output buses. The output F values are set during T1 and T2, the only requirement being that they be set prior to the T-step that gates their outputs. The ALU and SHIFTER outputs are stable at the end of T3, where both are simultaneously gated. If a look-ahead microinstruction fetch was in use, in place of FETCH, the entire instruction could complete execution in only 3 T-periods.

The ALU may be used for a full assortment of arithmetic and logical operations. Most arithmetic operations require use of carry-in and carry-out logic, while logical operations normally do not. In standard 18 bit, two's complement addition and subtraction the ALU carry-in must be cleared or set, respectively. Use of the opposite carry-in will result in a value one greater than the correct result for add operations, and one less for subtract operations. This may be used to advantage in multiple precision operations where low order additive carry-outs, or subtractive borrows, require corrective actions in the higher order results.

The next example is a simple single precision add. The carry-in hold must be cleared before the ALU will begin to compute a correct two's complement addition.

```
        ADD  A,B      [ R(A)=R(A)+R(B) ]
ADD:    "ADD LOCAL STORE REGISTER A TO B, RESULT IN A"
....  FETCH, KALC = ADD,  ALU STATUS ENABLE
X...  A->FAIL, B->FAIR, A->FADD, CLEAR CIH
.X..
..X.  GATE ALU, ALU TO COH
...X
```

All inputs are defined, including the carry-in-hold, in T1. The ALU results are available for gating at the end of T3. The K-Vector bit "ALU STATUS ENABLE" will normally be used in arithmetic and logical microinstructions, permitting the ALU status bits in FIST to be updated upon GATE ALU. This allows future instructions to test the results of this operation. The ALU carry-out must be manually transfered to FIST through the ALU TO COH command, which also saves that condition bit in the carry-out-hold register. If this instruction is a low order component of a multiple precision add then carry-out equal to 1 will indicate that the result was a 19 bit value, and will require an addition of 1 to the next higher order element. As stated above, this extra 1 addition may be accomplished during the next add operation by setting the carry-in-hold.

The next example illustrates a double precision subtract instruction. DSUB will subtract one pair of local store registers from another. Unlike some conventional double precision instructions, where the register numbers in the instruction operands point to the high order registers, our instruction operands point to the low order registers (for simplification of example). In this example the propagation of the carry-out from the lower precision result to the carry-in of the higher precision operation is accomplished through an internal carry command, "ALU TO BOTH CIH AND COH".

```
        DSUB  A,B      [ R(A-1).R(A)=R(A-1).R(A)-R(B-1).R(B) ]
DSUB:   "SUBTRACT REGISTER PAIR B FROM REGISTER PAIR A"
....  FETCH, ALU STATUS ENABLE, KALC = SUB, KA = RIGHT CTL
X...  A->FAIL,    B->FAIR,    A->FAOD,    SET CIH
.S..  DECF->FAIL, DECF->FAIR, A->FSID,    GATE ALU,   ALU TO BOTH CIH AND COH
..S.              KA->KSHC,   DECF->FAOD
...X                                      GATE ALU,   ALU TO COH
```

T1 sets all ALU local store pointers and the initial carry-in-hold as required for a subtract operation. The low order subtract proceeds during T2. At the end of T2 the low order result is gated, the carry-out required for the higher order subtract is passed back to the carry-in-hold, and the ALU inputs are decremented by one to point to the high order data local store registers. T3 is now used to decrement the ALU output pointer, and to set the SHIFTER control to right-control mode. This shift mode, along with FSID pointing to the low order result register (set in T2), will enable proper detection of the 35 bit "RESULT ZERO" condition. Finally T4 gates the high order result and sets the last carry-out indication into the carry-out-hold. Upon completion of this nanoprogram the CARRY, OVERFLOW, and SIGN bits of FIST accurately depict the final value of the double precision operation. The FIST "RESULT" bit shows the condition of the low order 35 bits of the AOD and SOD buses, as detected by recycling the low order result through the SHIFTER during the GATE ALU.

The INDEX ALU operates without the requirement for any associated F register to be preset. All INDEX actions must be stated, and completed, within two consecutive T-Periods. The following example modifies the LDCSR instruction, see section 7.3, to access a control store register relative to a base address instead of using an absolute address. The base address is maintained in an X register named X.BASE.

```
        LDCSRB  A,B      [ R(A)=CS(B+X(BASE)) ]
LDCSRB: "LOAD REGISTER FROM CONTROL STORE REGISTER PLUS BASE"
....  FETCH, KB = X.BASE, KX = 31., X KB = 3"CONSTANT INDEX REG SELECTION"
X...  A->FCOD, KX->FCIA, FIPH->A
.S..  INDEX (ADD), INDEX REG (X KB) = KX, G(G KX)
..X.
...S  READ CS (CIA), GATE CS
```

The final address consists of the original B field content plus the base address value from X.BASE. This address is ready at the end of T2, and may be used by the READ CS nanoprimitive at the beginning of T4. In this example the

ADD operation is commanded directly within the FSEL2 field of T-Vector T2. Optionally, the FSEL2 field may be used as a pointer to a command in one of the F registers. The index register is pointed to by the content of the KB field of the K-Vector, and the local store register is pointed to by the KX field. These register selections may only be made through indirect pointers, unlike the command selection field.

The SHIFTER may be used for either single length (18 bit) or double length (36 bit) shift operations. When used for single length shifts only the SID and SOD busses are used, with the KSHC and KSHA fields specifying the type of shift and shift amount, respectively. Double length shifts require use of the SID bus and one or both ALU busses (AIL and AIR) for input, and the SOD and AOD busses for output. If the ALU is not used for a combined ALU function and shift operation it may essentially be bypassed through use of the PASS LEFT ALU function. This allows the double length shift to complete in 2 T-periods. Use of most other ALU commands requires an extra T-period for completion of the ALU phase of the operation, before its output is ready for the high order action of the SHIFTER. The following example performs a double length shift, where the ALU is used strictly to pass the high order 18 bits to the shifter extension.

```
        SRDAI  A,B       [ R(A).R(A+1)=R(A).R(A+1)->B ]
SRDAI:  "SHIFT RIGHT DOUBLE ARITHMETIC IMMEDIATE"
....   FETCH, KSHC=RIGHT+DOUBLE+ARITHMETIC+RIGHT CTL, SH STATUS ENABLE
              KALC=PASS LEFT,                          ALU STATUS ENABLE
S...   A->FAIL,    A->FSID,    B->KSHA, CLEAR CIH
.S..   A->FAOD, INCF->FSID,    A->FSOD
..S.                        INCF->FSOD
...X   GATE ALU,   GATE SH, ALU TO COH
```

In a similar fashion to the ADD example above, the K-Vector SH STATUS ENABLE bit allows FIST to be updated at GATE SH time with the SHIFTER high bit and low bit status. These bits though are strictly the outputs of bits 0 and 17 on the SOD bus. The ALU STATUS ENABLE allows the setting of the SIGN, CARRY, OVERFLOW, and RESULT bits of FIST, upon GATE ALU execution. In the case of a right shift operation OVERFLOW will always be zero. CARRY should also be zero, and the CLEAR CIH operation in T1 will guarantee this. SIGN will be set to the correct 36 bit resultant sign. The RESULT bit of FIST will be set zero if all of the low order 35 bits (0 through 34) on the AOD and SOD busses are zero, as detected by "RIGHT CTL" mode.

This instruction is defined to shift the registers A and A+1. Therefore the low order shift operations will require that FSID and FSOD be set one greater than the value in the A field. This is accomplished by tranferring the value of A to all required F registers, and then incrementing that value directly in FSID and FSOD with the 6 bit increment functions in T2 and T3 respectively. Both inputs are stable immediately after T2, which would allow the results to be gated in a stretched T3 or at any time after T3. Faster techniques for setting up the final F register values may be worked out, with the INDEX ALU for example, reducing the overall time require for the SRDAI instruction.

As discussed above, when the ALU and SHIFTER are used for a combined function the time required for the passage of data through both the ALU and SHIFTER extension is 3 T-periods. An example of a combined operation where this capability may be used is in the extraction of subfields of data words. To extract a subfield and right justify the extracted result in a register we need several pieces of information. First the actual data source and the destination registers must be identified. Second the actual subfield width and position must be described in some form. For our next example we will simplify these items as follows. The source and destination data registers will be the same, as identified by the A field of our microinstruction. The actual original field width and situation will be found as a predefined mask in the local store register symbolically named LS.MASK. Finally, the right justification will be described by a shift amount specified as the micro-instruction B field.

```
          MASK.SHIFT  A,B      [ R(A)=(R(A)&R(LS.MASK))->B ]
MASK.SHIFT:  "MASKED-SHIFT REGISTER"
....  FETCH, KSHC = RIGHT+DOUBLE+LOGICAL, KALC = AND, KA = LS.MASK
S...  KA->FAIR, A->FAIL, B->KSHA, CLEAR COH
.X..  A->FAOD
..X.
...X  GATE ALU
```

Although the SHIFTER is actually involved in our instruction operation only the high order input and output component affect our result. Therefore FSID and FSOD are not referenced, nor is a GATE SH needed. The ALU inputs are set up in T1. The right input points to the fixed register LS.MASK and the left input points to the A field selected input. The shift amount is also set in T1, permitting the entire operation to begin immediately following T1. The CLEAR COH primitive is necessary to avoid the propagation of an extra high order bit into the result during the logical shift operation. The ALU output setting may be made any time preceding the GATE ALU. The ALU functions as follows. First the logical product of the content of register A and the mask register is computed. Then that result is shifted right, with left zero fill, and placed onto the ADD bus. This normally two step operation is now complete in one extended ALU-SHIFTER operation.

7.6 MULTIWORD NANOPROGRAMS

When the algorithm for a nanoprogram requires more than 4 T-steps additional nanowords must be used. There are several alternative methods that may be used in transferring control between nanowords. Selection of the appropriate method requires knowledge of the programming conventions in use within the set of resident nanoprograms making up the active nanostore environment. The most simple nanostore environment would be one where there are no nanowords shared by different nanoprograms. In most nanoprogramming the programmer tries to identify all common sets of procedures. If there exists frequently repeated exit code, from several nanoprograms, it may be possible for all to use the same last nanowords (referred to in the future as common tail nanowords). In a similar manner, if there exists frequently repeated sets of nanocode within different nanoprograms then it may be possible to set up a common body nano-subroutine.

Initial entry into the first nanoword of a microinstruction initiated nanoprogram uses "LOAD NPC(CS)" to set the entry address into the Nanoprogram Counter. The NPC may be modified from other sources. The KN field of the current nanoword may be transferred into NPC, using LOAD NPC(KN). NPC may be incremented by one, for each use of LOAD NPC(SEQ). Control may also be transferred without modifying the current NPC, using the NANOBRANCH selection mechanism. In this case the KN field is used directly to read the next nanostore location.

An example of the entry / exit procedure for calling on a nano-subroutine may be to require the calling nanoprogram to set the return nanostore address into the NPC prior to enterring the subroutine. This may be accomplished by either incrementing the NPC, with LOAD NPC(SEQ), or by actually setting up the entry address of the next nanoprogram using the next microinstruction operation code, with LOAD NPC(CS). Transfer to the subroutine would then be via NANOBRANCH address selection, with the BRANCH(address) nanoprimitive. The subroutine itself may transfer to multiple nanowords using only the NANOBRANCH mechanism. This subroutine would exit by simply reading nanostore without any alteration of NPC. The location read will be that originally desired for the return, by the calling nanoprogram. For example:

```
        CALLER                 []          SUBROUTINE
                               []
        BRANCH(SUBROUTINE)     []
T(N)    LOAD NPC(SEQ)          []  T(M)    READ NS, GATE NS
T(N+1)  READ NS, GATE NS       []          "RETURNS TO CALLER + 1"
```

The following example defines the BALTNW (Branch And Link on True to Next Word) instruction. This instruction requires two nanowords. The technique used for nanostore address modification is increment NPC. The Branch and Link operation takes place only when at least one of the bits selected in the B field is also set in FIST. If none of the selected bits are set control proceeds to the next sequential instruction.

```
        BALTNW  A,V,B
   [ IF ( B .AND. FIST ) THEN R(A)=MPC+2, MPC=V, ELSE MPC=MPC+2 ]
BALTNW: "BRANCH AND LINK TO CS(V) ON TRUE"
....  LEGAL MICRO OP ENTRY, KB = LS.MPC
S...  B->KS,          KB->FSID,      MPC PLUS 2,      LOAD NPC (SEQ)
.S..  KB->FCOD,       A->FSOD,       READ CS (MPC+1), READ NS,  GATE NS (NOT S)
..S.  READ CS (MPC),  LOAD NPC (CS)
...S  READ NS,        GATE NS,       LOAD R31

:
....  ALLOW INTS
X...  GATE SH,        GATE CS
.S..  READ CS (COD),  LOAD NPC (CS)
..S.  READ NS,        GATE NS,       LOAD R31
```

In the first word, T1 and T2 set up the "S" test condition from the B field and execute the test. If the result indicates "false" then control stays within this word, fetching the next sequential microinstruction. Otherwise, control transfers to the next sequential nanoword where the return MPC address is saved in R(A) and MPC is set to the value of the second word of the BALTNW instruction itself. The SHIFTER is used to pass the old MPC to R(A). Control store output data bus is used to supply the address for the READ CS in T2 of word 2, and for the new MPC value via GATE CS in T1 of word 2.

The first READ NS in word 1 reads word 2. This address is selected in T1 via the LOAD NPC (SEQ). If not used, NPC is reloaded from the microinstruction selected nanostore address in T3.

Note that although it appears that the original MPC value is incremented by 2, in T1 of word 1, the READ CS (MPC+1) in T2 still uses the original MPC value. This occurs because MPC PLUS 2 is a trailing edge operation and READ CS is a leading edge operation. Data propagation delay times guarantee that the MPC value cannot change before the control store address is completely decoded.

The next example describes an unsigned, 18 bit multiply instruction. The algorithm used in this example uses the conventional repetitive addition technique. In this nanoprogram the actual multiplication is done in a single nanoword. One initialization word and one completion word are also defined. The ALU and SHIFTER are connected during the operation, and as each conditionally selected addition is performed the result is shifted right one place from the ALU toward the SHIFTER. Upon completion the product has replaced the multiplier and multiplicand, in their original registers. The following diagram shows the ALU-SHIFTER organization for this program.

```
                                                        ADD/PASS TEST BIT
                                                                  /\
               PRODUCT OUTPUT   (ADD)       PRODUCT///MULTIPLIER  ""
                           ^                      OUTPUT  (SOD)   ""
   =======   =========================   =========================
   !CARRY!   !                       !   !                      ""!
   ! OUT !-->!  SHIFTER LEFT HALF    !-->!  SHIFTER RIGHT HALF  ""!
   !HOLD.!   !                       !   !                      ""!
   =======   =========================   =========================
      ^         //  //  //  //  //  /---/            ^
      !        //  //  //  //  //  /---/    (SID) OFFSET MULTIPLIER INPUT
      !      =========================
      ! CARRY!                       !
      !<-----!  A L U  (ADD or PASS) !
         OUT !                       !
             =========================
                    ^    :    ^
   PRODUCT INPUT  (AIL)  :  (AIR)  MULTIPLICAND INPUT
```

The above illustration shows the actual ALU-SHIFTER connections during the second word execution, UMULT2, only. Following each addition operation the ALU carry-out is placed into the carry-out-hold register. This carry value is then transfered into the ADD high bit during the right shift by 1 operation. The decision whether to add the multiplicand into the current result or only to pass the current result through the ALU to the SHIFTER is made by testing the carry-out-hold register value after the shift operation. This value will be set from the value of the SHIFTER low bit (SOD bit 0) at the end of each cycle.

```
        UMULT  A,B  [ R(A).R(B)=R(A)*R(B) ]
UMULT:  "MULTIPLY A TIMES B"
....  LEGAL MICRO OP ENTRY, BRANCH (UMULT2), KA = ZERO, KB = LS.WORK,
      KT = 18., KX = PASS LEFT, KALC = PASS LEFT,
      KSHC = RIGHT+SINGLE+LOGICAL+RIGHT CTL
X...  A->FAIL,  KB->FADD, B->FSID,  CLEAR CIH
.S..  KB->FAIR, A->FADD,  B->FSOD,  GATE ALU,  READ CS (MPC+1), MPC PLUS 1,
                                               LOAD NPC (CS)
..S.  KA->KALC, KX->F.PASS,         SH TO COH, READ NS
...S  KT->F.COUNT,                  GATE ALU,  GATE NS

: UMULT2 = N.  "CONTINUATION OF UNSIGNED MULTIPLY"
....  BRANCH (N.+1), KALC = PASS LEFT, KSHC = RIGHT+DOUBLE+LOGICAL, KSHA = 1,
      KT = CARRY, KX = F ZERO, KB = ADD
S...  KB->KALC,   SKIP (NOT T)
.X..  F.PASS->KALC
..S.  ALU TO COH, READ NS, GATE NS (X), DECF->F.COUNT
...X  SH TO COH,  GATE ALU, GATE SH

:     "COMPLETION OF UNSIGNED MULTIPLY"
....  ALLOW INTS, KSHC = LEFT+DOUBLE+LOGICAL+RIGHT CTL, KALC = PASS LEFT
      ALU STATUS ENABLE
X...
.S..  ALU TO COH, GATE ALU, GATE SH, READ NS, GATE NS, LOAD R31
```

The following discussion will cover the action of each nanoword in the above example. In the initialization word, T1 prepares the ALU to pass the multiplicand to local store register LS.WORK for use as the multiplicand source during additions. Carry-in-hold is also cleared for the add operations that will follow. T2 executes the GATE ALU, saving the multiplicand. The ALU and SHIFTER bus connections are completed. All microinstruction fetch actions are completed: READ CS (MPC+1), LOAD NPC (CS), and MPC PLUS 1. The next nanoprogram address is now selected, but will not be referenced until the READ NS in the completion word. T3 changes the ALU function to ZERO, which will clear the initial product value in R(A). An F register named F.PASS is initialized with the value of the ALU pass function. COH is set to the value of the right most bit of the multiplier, taken from SID bit 0 using the "RIGHT CTL" function of the SHIFTER. Nanostore location UMULT2 is read. Finally, T4 sets an F register named F.COUNT to the value 18 (decimal), which will be used by UMULT2 as a counter during its 18 loops. Register R(A) is zeroed via the ALU, and control is transfered to T1 of UMULT2.

UMULT2 performs the actual multiplication. This requires it to repeat its full 4 T-steps 18 times. T1 is used to set the ALU control to ADD mode, and to determine whether to change the function to PASS LEFT. If the current right-most bit of the multiplier is a 1 then T2 will be skipped, leaving the ALU set for an add operation. If a 0 then T2 is executed and the ALU will not alter the result during this loop cycle. T3 transfers the ALU carry-out condition to the carry-out-hold, as required before the actual GATE ALU is performed, to preset carry-out-hold with the correct value to be propagated into the SIGN bit position of AOD. T3 also reads the next nanostore location, and makes the decision to terminate the multiplication when F.COUNT is decremented to zero. T4 now completes the cycle by gating the new partial product into R(A) and R(B) along with the shifting of the multiplier right one bit position. The SH TO COH operation sets COH to the value of the new right-most bit on SOD, to be used for the ADD/PASS decision in T1 for the next loop cycle.

The completion word is required only if it is desired to set FIST to accurately portray the SIGN and RESULT of the final 36 bit product. T1 is empty, allowing for interrupt address selection and for ALU-SHIFTER propagation. T2 gates the ALU and SHIFTER back into their current registers, unmodified, only to cause the correct setting of SIGN, RESULT, OVERFLOW and CARRY in FIST. CARRY and OVERFLOW are meaningless in an unsigned multiply operation.

Since UMULT2 is actually a complete multiply routine any nanoprogram requiring a multiplication operation as its last procedure may use it as a common tail. With minor alterations, UMULT2 may also be used as the final phase of a signed multiply routine.

8 QM-1 I/O SYSTEMS

8.1 GENERAL

The QM-1 has 8 independent I/O ports (see Section 4.6, External Interface). Any or all of these ports may be used for concurrent data transfers and device control operations. A QM-1 port may be interfaced directly to a user's own equipment or to NANODATA standard peripheral devices through the QM-1 Channel Controller. Refer to Appendix-A for a description of actual QM-1 port interfacing.

Figure 8.1A is a block diagram of the standard QM-1 I/O System. No more than one channel controller may be attached to each CPU port. Up to 64 standard device controllers may be connected to each channel controller. More than 64 actual I/O devices can be placed on a single channel due to many forms of device controllers supporting more than one device (ie. tape drives, telecommunications devices). For connection of user owned equipment to a QM-1 Channel Controller, refer to the document "NANODATA STANDARD CHANNEL CONTROLLER".

Data transfers are maintained on a word to word basis. All devices on the same channel may be transferring data simultaneously, as long as their combined data rates do not exceed the destination memory access speed. Up to 18 bits at a time are transferred between the port and device, over each active channel. When data is being passed directly to a CPU port the CPU is interrupted periodically, in order to route each datum between the I/O port and appropriate QM-1 memory. The standard QM-1 device controllers maintain data routing information for the duration of the full data block transfer. This consists of automatic storage and updating of the memory address pointer and block length word counter. In addition, the CPU is notified at the end of operation or of other programmably selected conditions; such as device errors, device ready state change, word count reaching zero, etc.

QM-1 systems equipped with the optional multiport main store interface may take advantage of the Direct Memory Access (DMA) path. With the addition of the appropriate DMA channel controllers, I/O data transfers may proceed without CPU intervention. In DMA operating mode the CPU is usually involved in the data transfer operation only at initiation, and following termination. This permits devices with high speed data transmission rates to be active without adding any direct CPU overhead. Low speed devices may also utilize the DMA path, at the installation's option.

The standard QM-1 multiport memory consists of up to 8 external ports. At least one of these ports must be connected to the normal CPU main storage access bus. An additional port is required for each CPU within a multi-processor environment. All remaining main store ports may be connected to individual DMA controllers, permitting up to seven DMA connections. When main store is configured as a four way interleaved memory (750 nanoseconds full cycle, 18 bit access) an aggregate data rate of over 10 million bytes per second may be realized.

The following sections describe the individual components of the QM-1 I/O System in full detail.. For a thorough understanding of QM-1 I/O it is recommended that the reader also be familiar with the QM-1 External Interrupt Mechanism (sections 4.5.2 and 4.5.4.2), External Store (section 4.2.5), and the External Interface organization (sections 4.6 and 5.4.4).

4.5.2.4

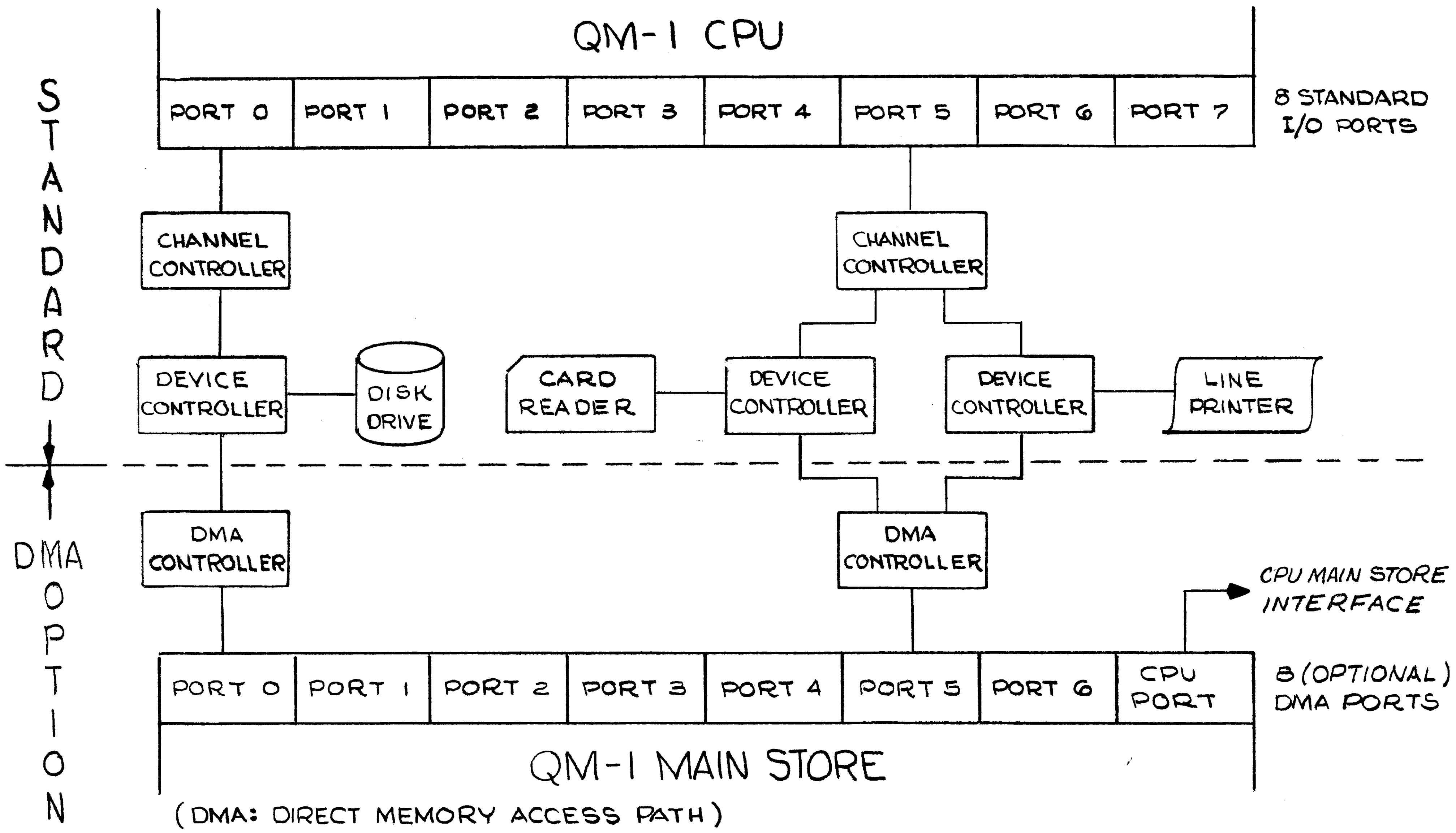


FIG 8.1A - QM-1 STANDARD I/O SYSTEM

8.2 QM-1 I/O OPERATION

8.2.1 GENERAL

The QM-1's I/O System, being consistent with the design of QM-1, consists of individual functions which may be utilized in numerous ways to move information into and out of the QM-1 CPU. The following section is a description of the relationships between the various I/O support functions and their overall operation relative to the rest of the CPU. Section 8.2.2 is concerned with the data routing support facilities of QM-1 I/O, available at the ports, while section 8.2.3 discusses the interrupt related operations.

8.2.2 QM-1 PORT OPERATION

Section 4.6 describes a QM-1 port and the facilities and controls available for I/O support. This section is therefore devoted to the relationship of these controls to each other and to overall CPU timing.

The first piece of information necessary for any port operation is "KA". The low three bits of "KA" select the port at which an operation will be performed. "KA" must be stable on the leading edge of the T-Period in which an I/O operation is to take place. The following program segments all try to send an XIO signal to port three (3). The first two are valid while the third fails.

```
1.      :
        ....   KA = 3   "SELECTS PORT 3"
        X...   XIO

2.      :      "ASSUME FLIV CONTAINS 3"
        X...   FLIV->KA
        .X..   "WAIT FOR PORT SELECTION"
        ..X.   XIO

3.      :      "ASSUME FLIV CONTAINS 3"
        X...   FLIV->KA
        .X..   XIO      "FAILS BECAUSE KA WAS NOT DEFINED IN TIME"
```

These three examples hold for RIO as well.

The relationship of loading a port register to an operation on the port is very important. The following program segment gates main store into port register 3 (External store 3 via FMOD = 32 + 3, see section 4.2.6) and then sends XIO to port 3. The data will be valid at the time of XIO.

```
:
....  KA = 3 "SELECTS PORT 3", KB = 43 "OCTAL, 35. DECIMAL"
X...  KB->FMOD "POINT MOD TO ES(3)"
.X..  GATE MS  "GATES MOD TO ES(3)"
..X.  XIO      "SENDS DATA TO DEVICE"
```

If XIO had been in T2 instead of T3 the old value in ES(3) would have been at the port when XIO occurred.

The same timing holds when the external store register is being loaded from local store via the LOAD ES nanoprimitive.

Data being read into the CPU has much the same timing as the LOAD ES function. The RIO primitive causes the data at the "KA" selected port to be loaded into the associated port register. This function is leading edge with "trailing edge results". This means that the transfer is started on the leading edge and completed on the trailing edge of the T-Period. This is important as shown in the following program segment.

```
:     "ASSUME FEOA = 3 AND FEOD = 1"
....  KA = 3
S...  RIO     "READ THE DATA FROM PORT 3 INTO PORT REGISTER 3, ES(3)"
S...  GATE ES "GATE THE SAME DATA INTO LOCAL STORE REGISTER 1"
```

The GATE ES is valid, since the RIO was completed by the trailing edge of the first T-Period of the stretched T-Step. The GATE ES then took place at the trailing edge of the second T-Period of the same T-Step.

The 6 bit transfer primitive "IO ID -> F REG" is a standard trailing edge function which gates the contents of the I/O ID lines of the "KA" selected port into the specified F-Register. Timing is the same as XIO and RIO with respect to "KA" port selection, however it is a true trailing edge function while XIO and RIO are leading edge with trailing edge timing.

The two six bit buses from the QM-1 to the ports are the "G-Bus" and the "Phantom-Bus" (see section 4.6). All following text will refer to these busses as the "Device Selection" and "CPU Command" busses respectively. These terms represent suggested uses of these two six bit data paths, and in no way limits their use for any other purpose. NANODATA Standard QM-1 I/O equipment uses the six bits of the "G-Bus" to select one of sixty four (64) device controllers on a channel while the "Phantom-Bus" is used to specify one of sixty four (64) Commands to be performed when an XIO signal is sent to a channel.

The timing of the Command and Selection lines is such that they are stable at XIO time if they are specified within the same T-Step as the XIO signal. The following program segment places the value "4" on the Command lines, the value "75" on the Device Select lines, and sends XIO to port 3.

```
:
....  KA = 3   "PORT NUMBER", KS = 75, KX = 4
X...  G(G KS)  "VALUE OF KS (=75) OUTPUT ON DEVICE SELECT LINES"
      KX->FIPH "VALUE OF KX (=04) OUTPUT ON CPU COMMAND LINES"
      XIO      "XIO IS SENT TO PORT 3"
```

Section 4.6 describes a number of signals used for timing and interrupt purposes as well as a special signal called "Master Clear". The Master Clear signal is generated when the CPU is powered on, or when the "System Reset" button is depressed. The signal is available at the port so any external devices may be initialized along with the CPU.

8.2.3 QM-1 INTERRUPT OPERATION

This section discusses the QM-1 interrupt system in terms of program timing and sequences rather than the actual hardware units involved, hence a thorough understanding of sections 4.5.2 and 5.8.2 is necessary before going further.

The starting point for an external QM-1 interrupt is an interrupt from some external source. The interrupt is now in the latched state. There is no way, within the CPU, to prevent an interrupt from latching. Once latched, the interrupt remains latched until acknowledged in the normal manner or cleared by the Generate/Clear Interrupt mechanism (see section 5.8.1) or Master Clear. The latched interrupt must be enabled by the programmer in order to become pending. A latched interrupt becomes pending following the first GATE NS primitive after the corresponding enable bit is set. Similarly, an enabled interrupt becomes pending on the first GATE NS after it is latched. If an interrupt is pending when its corresponding enable bit is turned off the interrupt remains pending until the next GATE NS. This means that the nanoword which clears any enable bits should not allow interrupts!

Once an interrupt is pending it becomes available for priority selection. As mentioned in section 4.5.2, the priority selection mechanisms are activated/deactivated by the ALLOW NANO and ALLOW MICRO interrupt nanoprimitives. A timing conflict arises when READ NS (of NPC) occurs in T1 and either Allow interrupt control is also specified. The result of the READ NS is undefined since the read is leading edge and the priority mechanism was activated on the same leading edge. The obvious rule of thumb is not to READ NS (of NPC) in T1 of any nanoword which also allows interrupts.

The overall timing of an interrupt, from latching to acknowledgement, is a function of the executing nanocode. The three operations necessary to have a latched interrupt acknowledged (assuming it is enabled and allowed) are:

1.	GATE NS	Causes the interrupt to become pending.
2.	READ NS (of NPC)	Reads the interrupt selected nanocode.
3.	GATE NS	Gates the interrupt nanocode and clears the interrupt latch and pending flags.

An interrupt may also be cleared by Generate/Clear Interrupt (section 5.8.1). The Clear Interrupt function clears both the latched and pending flags of an interrupt. The function is completed by the trailing edge of T1.

8.3 STANDARD CHANNEL CONTROLLER

8.3.1 GENERAL

The behavior of the standard QM-1 I/O Channel is similar to that known throughout the computer industry as a "Multiplexor" channel. This means that several independent I/O devices may be concurrently transferring data over the same I/O channel, without knowledge of each others existence. Of major interest at this point is the fact that unlike a conventional multiplexor channel, which supports either low speed devices only or high speed devices when the low speed devices are inactive, the QM-1 standard channel will support an active mixture of both classes of devices. This is feasible as long as their aggregate data rates do not exceed the access speed of the destination memory. For example, a high speed disk transfer of 800,000 bytes per second, a tape drive transfer of 90,000 bytes per second, and several low speed unit record devices (ie. printers, punches) could be active together over the same DMA path to main store without encountering loss of data due to interference within the channel. Of course the system designer should be aware that there is always the hazard of lost data due to one or more of the desired memory banks being tied up through activity from other storage ports. The above mixture warrants being assigned to the highest priority main store port.

The QM-1 Channel Controller coordinates the communication between the QM-1 CPU and I/O device controllers, and synchronizes the demands for data and status interrupts to the CPU. The relationship of the channel controller to the other elements of the I/O system is shown in figure 8.3.1A. Whenever the CPU sends an XIO signal the controller will either respond to the command or will immediately pass the command to the device controller indicated. I/O commands are divided into two categories, channel commands and device commands. Each command is recognized as a six bit quantity, found on the CPU command lines during an XIO signal (command lines originate on the Phantom Bus, see section 8.2.2). Commands with values in the range of 00 - 67 (octal) are considered device commands and are passed directly to the device controller identified by the six bit quantity on the device select lines (originating from the G Bus, see section 8.2.2). Commands with values 70 - 77 (octal) are channel commands and result in immediate channel controller action. The specific channel commands are discussed in section 8.3.2.

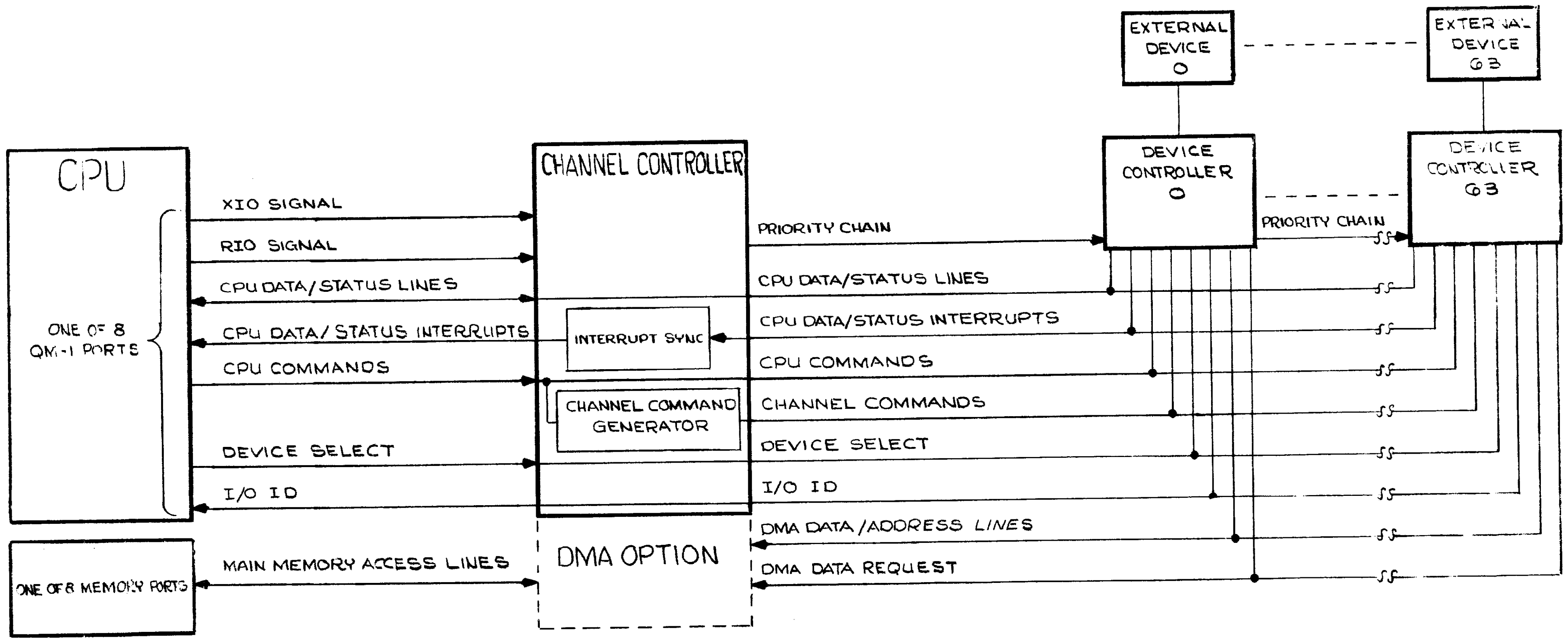


FIG 8.3.1A - CHANNEL CONTROLLER

The channel may be considered to have three modes of operation: direct request, data, and status. The actual mode of operation is maintained jointly by the channel controller and each device controller (discussed in section 8.4). Direct request mode is a momentary channel state which exists between the time of issuance of a direct request command (via XIO) and the gating of the desired datum (via RIO). An example of an actual direct request command is "read word count" register. This command is transmitted to a specific device controller. The channel controller locks itself into this operation until an RIO signal is received from the CPU, gating the requested information into the appropriate I/O port (ES) register. It is the responsibility of the requesting CPU program to allow for the propagation time of the original command to reach the specified device controller and for the desired information to return to the port data lines. For standard I/O cable lengths (up to 75 feet from the CPU to the channel and device controllers) a 300 nanosecond round trip is required. The following is an example of a direct request procedure, followed by a timing diagram.

```
GET STATUS:                  "FROM CHANNEL A, DEVICE B"
....    KA = ** "PORT SELECTION", KB = RD.WORDCT "STATUS COMMAND VALUE"
....    HOLD 2  "RETAIN KA AND KB", BRANCH( READ PORT ) "NEXT NANOWORD"
S...    A->KA                "SET PORT NUMBER INTO PORT SELECT (KA)"
.X..                         "PORT SELECTION SETTLING TIME"
..S.    G(G B)               "DEVICE SELECTION NUMBER FROM B PARAMETER OF R31"
..S.    KB->FIPH             "COMMAND VALUE TO COMMAND LINES"
..S.    XIO                  "INITIATE COMMAND TRANSMISSION"
        "AT LEAST 5 T-PERIODS SHOULD ELAPSE FROM LEADING EDGE OF XIO"
...S    READ NS,  GATE NS

READ PORT:                   "PORT NUMBER PASSED VIA KA, USING HOLD 2"
....
X...    READ NS
.X..    RIO                  "PLACE DESIRED COUNTER INTO PORT REGISTER"
.X..    GATE NS              "EXIT THIS PROCEDURE"
```

** Indicates an undefined value, to be set up during program execution.

```
                       :     ----           :
          I            :    I    I          :
    XIO   I_________________I    I_____________________________________
          I            :                    :              ----
          I            :                    :             I    I
    RIO   I_______________________________________________I    I_______
          I            :                    :
 T-PERIOD I  1  /  2  /: 3  /  4  /  5  /  6: /  7  /  8  /  9  /
                       :                    :
                       \_SET PORT SELECT.   \_DEVICE CONTROLLER
                                              TRANSMITS STATUS REGISTER.
```

While direct request mode is initiated by the CPU, both data and status modes are selected from demands of the device controllers. The actual notification at the CPU, of channel entry into one of these modes is through data or status interrupts. A status interrupt originates at an I/O device, and indicates either a change of state within the device or an end of operation condition. All conditions capable of triggering status interrupts may be masked, under programmed control, as to whether their occurrence may request the interrupt or only set a status indicator. The channel controller is continually interrogating its device controllers for the presence of a status interrupt request flag. This interrogation proceeds from the channel controller, down the priority chain line, through each device controller, in the order of their connection on the priority chain. Once a status interrupt request flag is recognized the channel controller locks out further status request analysis and transmits a status interrupt to the CPU. At this time the device identification (IO ID) of the interrupting device, and its interrupt status register, is retained until called for by the CPU. No further status requests will be accepted by the channel controller until the CPU has completed the handling of this interrupt.

The issuance of a status interrupt to the CPU prepares the channel to enter status mode. This mode is not actually entered until the CPU processes the status interrupt. Status mode may be considered a momentary channel state, as described for direct request mode above. It is entered following issuance of the channel command "Status Request" (described in section 8.3.2) and is terminated with the next RIO signal. Timing is the same as for direct request (above). The status interrupt, "Status Request", sequence is the only method provided for accessing an Interrupt Status Register (see section 8.4.2.2).

Data interrupt processing is handled independently of status interrupts. Using the same priority ordered interrogation procedure defined for status interrupt requests (above), the channel controller searches for the occurrence of a data interrupt request. This interrogation is not affected by the presence of a pending status interrupt, nor will the interrogation for status requests be affected by a pending data interrupt. As soon as a data interrupt request is recognized the channel is placed immediately into data mode. The search for any other data interrupt requests is inhibited until the CPU has processed this data interrupt. The channel remains in data mode until the active datum is passed between the CPU and I/O device. Prior to the actual processing of a data interrupt, data mode may be temporarily suppressed by CPU action to place the channel into either direct request or status mode. When the momentary channel operation is completed the channel will return to its previous data state.

Data interrupt processing varies, depending on the direction of data flow requested. A device controller processing an output operation will make data-out interrupt requests of the channel, while input operations will make data-in interrupt requests. When the channel controller is processing a data-in interrupt it will remain in data mode until exactly two (2) RIO signals have been sent by the CPU. The first RIO indicates that the CPU is beginning to handle the current interrupt. The information present on the port data lines, since entering data mode, is the memory address at which the following data word is to be stored. This address originates from a register within the device controller involved in this data mode operation, and is described in section 8.4.3 (Data Routing Support). The RIO gates the address into the port register and then allows the channel controller to fetch the actual data word from the device controller. The second RIO gates the data word into the port register, and releases the channel controller from data mode. Interrogation for the next data interrupt request can now begin. The CPU program handling the input data operation must allow for propagation time between the two RIO signals. It takes approximately 300 nanoseconds for the first signal to reach the device controller and for the actual data to return to the port data lines (75 foot cable lengths).

Data-out interrupt processing differs slightly from data-in operation. Exactly two signals are required to release the channel from data mode. In this case an RIO signal will gate a memory address into the port register, as with data-in above, and as soon as the specified data word has been placed into the port register the channel command "Data Available" (described in section 8.3.2) is issued with an XIO signal. This command transmits the datum to the I/O device and releases data mode.

During status mode and data mode operations the CPU need not provide device selection information on the G-Bus lines. This is due to the channel controller unconditionally locking into the device controller associated with the interrupt. On the other hand, the CPU usually does need to know the identity of the device causing an interrupt. For this reason each device controller will present its unique device selection number to the channel's "IO ID" lines during status mode. The correct "IO ID" will be available 300 nanoseconds after the issuance of the "Status Request" command, and will remain stable until after the RIO release signal. During data mode interrupts the device ID is normally unnecessary, since the presence of the memory address is all that is needed to complete the data transfer cycle. When using QM-1 virtual memory options an additional identifier, called the "task ID", is required to select the appropriate storage partition. The "task ID" replaces the device ID on the "IO ID" lines, and is originally preset by the Operating System under programmed control (see section 8.4.2.1, Device Control Word).

8.3.2 CHANNEL COMMANDS

Seven of the eight possible channel commands are currently assigned. Their function and usage are described below. Command code values are shown in octal. Transmission of a channel command requires placement of the command value on the Phantom bus lines during issuance of the XIO signal. The command is then sent to the channel controller on the "KA" selected I/O port.

I/O RESET (70) - Resets the Channel Controller and all Device Controllers on that channel. Each controller is set to its initial state. All internal registers are cleared to zero, and all interrupt flags and status indicators are reset. This command should be used only during system initialization.

DATA AVAILABLE (71) - Used only during DATA-OUT mode channel operation, following the memory address gate RIO signal. Data Available is issued following placement of the actual data into the appropriate port register. DATA-OUT mode is cleared immediately following this command.

(72) - Unassigned at present.

ENABLE DATA INTERRUPTS (73) - Allows data interrupt requests to be generated at the device controllers and to be issued to the CPU. Since initial channel conditions inhibit interrupts this command must be used prior to data transfer operations; following a "Disable Data Interrupts" command (74), an "I/O Reset" command (70), or System "Master Clear" manual operation.

DISABLE DATA INTERRUPTS (74) - Blocks the posting of data interrupts by the device controllers. This is an initial state condition (I/O Reset), and may be cleared with an "Enable Data Interrupts" command (73).

ENABLE STATUS INTERRUPTS (75) - Allows status interrupt requests to be accepted by the Channel Controller and to be issued to the CPU. Since initial channel conditions inhibit interrupts this command must be used prior to normal channel activity; following a "Disable Status Interrupts" command (76), an "I/O Reset" command (70), or a System "Master Clear" manual operation.

DISABLE STATUS INTERRUPTS (76) - Blocks the selection of status interrupts by the Channel Controller. This is an initial state condition (I/O Reset), and may be cleared with an "Enable Status Interrupts" command (75).

STATUS REQUEST (77) - Used only following a status interrupt, in order to place the channel into STATUS mode. Following issuance of this command the interrupting Device Controller places its interrupt status register onto the port data lines (up to 18 bits of information) and its device identification on the "IO ID" lines (6 bits). Following appropriate command/data propagation delay (usually 300 nanoseconds) an RIO signal will gate the interrupt status register data into the I/O port register and release the channel from status mode.

8.4 STANDARD DEVICE CONTROLLER

8.4.1 GENERAL

The standard device controller maintains control over one or more physical I/O units, and acts as the interface between these units and the standard channel controller. The device controller is designed to allow generalized software to handle a vast array of very different I/O equipment. It contains a number of features which may, or may not, be used by the software, allowing a varying degree of hardware support. Figure 8.4.1A shows the relationship of the device controller to the standard CPU, Direct Memory Access channels and the external I/O devices. This figure also shows the major hardware features of the device controller. These features are divided into two major categories; device control support and data transfer support.

Within the device controller some of the control support units can be classified as "static controls". These are the "device control words", the "status register", and the "interrupt status register". These are referred to as "static" controls since they all contain residual information to be used by the rest of the I/O system, usually beyond the scope of one device operation.

The remaining control support units are classified "dynamic controls". These are the units which respond directly to commands from the CPU, the channel controller or the device itself.

Additional units are provided for data routing support. These are the Buffer Address Register (BAR), which serves as a memory address pointer, and the Word Count register (WC), which can be used to count the number of words, or characters, to be transferred. WC can be decremented and BAR can be incremented or decremented automatically during data transfers. These registers may be applied to data transfers over the data path to the actual QM-1 port register, or directly to main store when the Direct Memory Access option is installed in the channel controller. Finally, a "data chaining" facility is provided which allows the software to combine disjoint blocks of memory into one logically continuous I/O buffer, with minimal software intervention. The device controller also contains the hardware necessary to recognize and process interrupt requests from the device, as well as to generate special interrupts on device controller status or DMA channel status conditions.

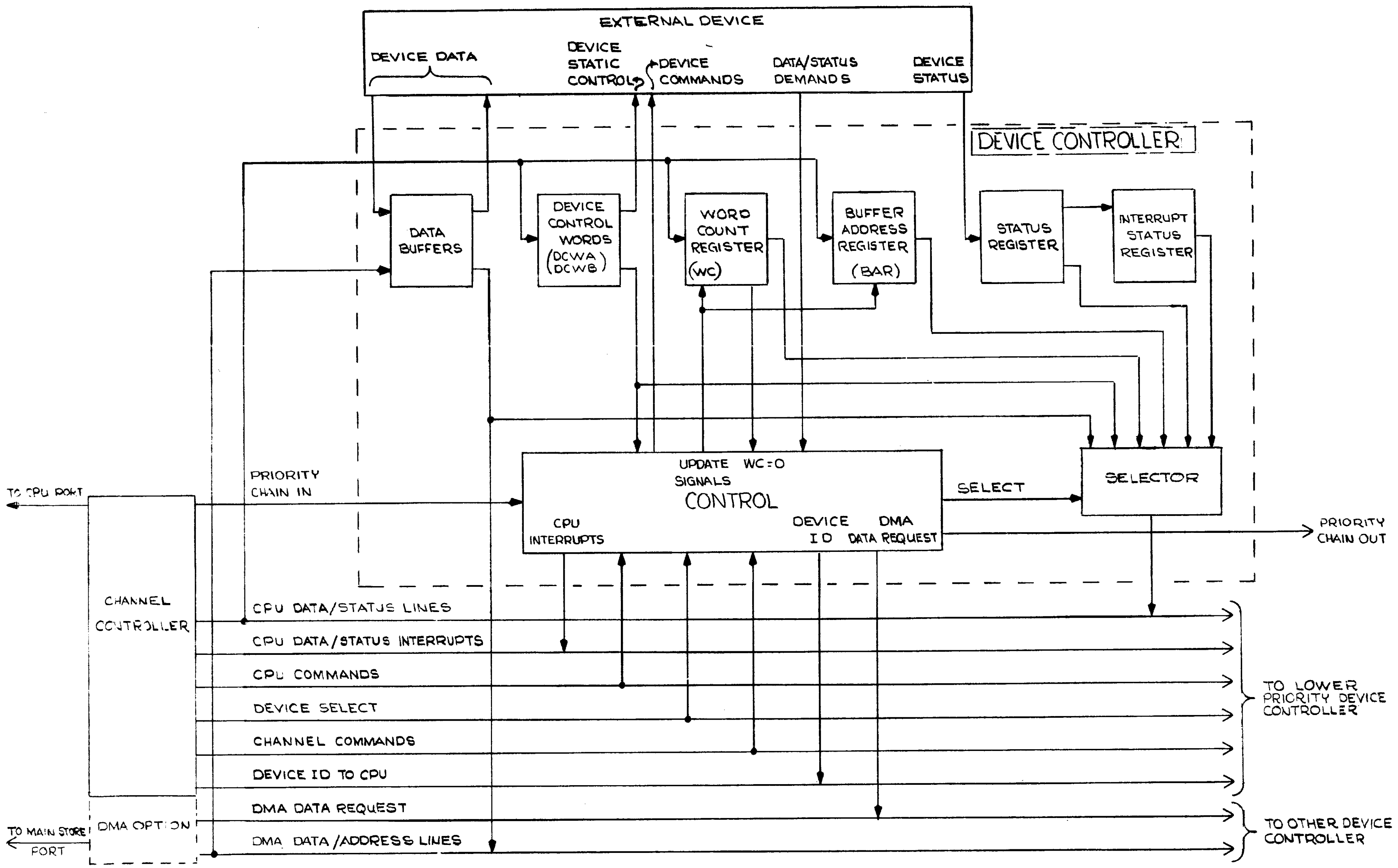


FIG 8.4.1A-DEVICE CONTROLLER

8.4.2 DEVICE CONTROL

8.4.2.1 DEVICE CONTROL WORDS

The Device Control Words consist of two registers containing "static" control information, such as "select DMA operating mode", "activate data translation", etc. The two registers are referred to as DCWA and DCWB. Their content varies depending on the devices attached to the controller. The DCW allows the programmer to select the functional units of the device controller that will be utilized in an operation. For example: DCWA allows the programmer to select events on which status interrupts will occur.

DCWA and DCWB are loaded by specific device commands (see sections 8.4.2.5 and 8.5.3), and are accessible to the programmer at any time via the direct request mechanism (sections 8.3.1, 8.4.2.4, and 8.5.6). NANODATA has assigned standard (device independant) functions to several DCWA and DCWB bit positions, that are common to a wide range of external devices. The following lists describe their functions.

Bit Identity	Description
DATA MAPPING CONTROLS	
DCWA 02 TRANSLATE	_Controls character code translation within the device controller. This feature is optional in some devices. Example: Translate lower case ASCII to upper case ASCII for printing.
DCWA 03 PACK/UNPACK	_Controls the mapping of characters (bytes) into the standard QM-1 18 bit word. In "packed" operating mode most device controllers treat the WC register as a character counter, and decrement by two for each QM-1 word transferred.
STATUS INTERRUPT ENABLE MASKS	
DCWA 04 COMMAND REJECT	_See sections 8.4.2.2 and 8.4.2.5 for complete information on "command reject".

DCWA 05	ANY ERROR CONDITION	_See section 8.4.2.2 for a description of the "error" status bit controls.
DCWA 06	DEVICE READY	_Allows interrupt on device state changing from ready to not ready, or not ready to ready, condition. See section 8.4.2.2.
DCWA 07	NOT BUSY	_Allows interrupt on device state changing from busy (performing an operation) to not busy (device inactive). See section 8.4.2.2 for additional discussion.
DCWA 08	UNIT AVAILABLE	_Allows interrupt to indicate that the Device is ready to receive commands for its next operating cycle. See section 8.4.2.2.
DCWA 09	WORD COUNT ZERO	_Allows interrupt on word count register being decremented to zero. Terminates current I/O transfer. See 8.4.2.2.

IDENTIFICATION

DCWB 00	TASK ID 0	_Task ID is simply a six bit field, the content of which is placed onto the device ID lines during data interrupts, by the device controller. Its purpose is to provide additional data routing information for the system utilizing the device. See section 8.4.3.3 for additional details.
DCWB 01	" " 1	
DCWB 02	" " 2	
DCWB 03	" " 3	
DCWB 04	" " 4	
DCWB 05	" " 5	

DATA ROUTING SUPPORT CONTROLS

DCWB 06	DECREMENT WORD COUNT (WC)	_Permits the content of the WC register to be decremented following each data transfer cycle. See sections 8.4.3.2 and 8.5.4.
DCWB 07	INCREMENT BUFFER ADDRESS REGISTER (BAR)	_Permits the content of BAR to be incremented following each word transfer cycle. See sections 8.4.3.1 and 8.5.4
DCWB 08	DECREMENT BUFFER ADDRESS REGISTER (BAR)	_Same as DCWB 07, except decrements register value. See sections 8.4.3.1 and 8.5.4
DCWB 09	DMA MODE	_Switches the device to DMA operating mode. DMA option is required. See 8.4.3.5.

NOTE: DCWA and DCWB are cleared (zeroed) by Master Clear, I/O reset, and Clear Device commands.

8.4.2.2 STATUS

Each device controller is provided with two status registers that inform the I/O processing program of the state of the device and its associated controller. These two registers are called the "Status Register" (SR) and the "Interrupt Status Register" (ISR). The Status Register contains full and detailed information on the condition of the device and controller. The Status Register is accessible to the program at any time via the "direct request" mechanism (described in sections 8.3.1 and 8.4.2.4).

The Interrupt Status Register (ISR) differs from the Status Register (SR) in the nature of its bit content and operation. The ISR contains a more sparse form of SR information bits. One bit of ISR may represent a class of bits in SR. For example, the general "error" bit of ISR represents all error conditions of the device and device controller: lost data, bad parity, etc. In a complicated device, where more than one Status Register may be required, ISR is used as a pointer to the Status Register that contains the detailed information on the reason for the status interrupt. The primary purpose of the ISR is to inform the I/O process of which status condition(s) caused the status interrupt currently being handled (such as normal end of operation, a device error, etc.)

The setting of an ISR bit always occurs along with a status interrupt. In order for an ISR bit to set it must first be enabled by the prior setting of its corresponding DCW bit (see section 8.4.2.1, device control word). The setting of an ISR bit occurs only during change of the related state, while the condition is enabled. For example, the device "not busy" bit will set, and generate a status interrupt, only on the change of device state from "busy" to "not busy" while DCWA bit 07 (enable interrupt on "not busy") is set. If the device is already not busy when, bit 07 of DCWA is changed to enable the interrupt (made a one), no interrupt will occur. The device must change state from "busy" to "not busy", while the interrupt is enabled, to trigger the interrupt.

In order for the active I/O processing program to read the ISR it must issue a "status request" command. It may issue this command only after the receipt of a status interrupt. Since the channel controller "remembers" the identity of the device controller currently requesting the status interrupt, the "status request" command will be routed directly to that device. This function is described in more detail in the following section on interrupts. Upon receipt of a "status request" the device controller will put the content of its ISR, and its device selection number, on the channel data, and device ID, lines to the CPU. A time delay of 300 nanoseconds is required between the "status request" command and the RIO that reads the status data into the port register. The device controller does not allow any additional ISR bits to set following its receipt of the "status request", until completion of status interrupt processing by the next RIO. All ISR bits set, and their corresponding interrupt requests, prior to a "status request" will be considered as fully acknowledged following the response to just one status interrupt. Therefore several enabled status interrupt conditions, occurring within close proximity of each other, can be processed as a single status interrupt. All ISR bits set prior to the interrupt will be cleared automatically following the transfer of the ISR to the CPU.

There are three methods by which status bits may be reset (cleared or zeroed). The first type of status information may be referred to as "real time". This means that the device condition itself has control over the state of the status bit. "Device ready", for example, belongs to the "real time" category, since it will change state every time the device ready condition changes. The second type of status are the "error" condition bits. They are cleared through issuance of the "clear error" command, which is described in the section on device commands (8.4.2.5 below). Finally, the status bits which are related to word count going zero are cleared by the loading of a new value into the Word Count register. All status bits, except for "real time", are cleared with "Master Clear", "I/O Reset" (see section 8.3.2, channel commands), "Clear device" (8.4.2.5), as well as "Clear error" (8.4.2.5).

The positions of most ISR bits correspond to those bits within SR having the same general meaning. For this reason only one status bit description list appears below. All status indicators described appear in the same postitions in both ISR and SR. The following list specifies standard Interrupt Status, and Status, register bits assigned by NANODATA to cover a wide range of external devices. Bit position numbers are shown in decimal. All ISR indicators, described below, will be set only if enabled in the corresponding DCW.

BIT POSITION & IDENTITY	DESCRIPTION OF STATUS
0 - DEVICE READY	(Real time controlled in SR) "ISR" Status interrupt generated, and bit is set, whenever a change occurs in the device ready condition. "SR" A 1 indicates that the device is currently "ready", 0 indicates "not ready".

1 - DEVICE NOT BUSY	_(Real time controlled in SR) "ISR" Status interrupt generated, and bit is set, when the device changes state from "busy" to "not busy". "SR" A 1 indicates device currently "not busy", 0 indicates device in operation.
2 - ERROR CONDITION	_"ISR" Status interrupt generated, and bit is set, whenever any error condition occurs in the device or controller. Error conditions setting this bit include any of those listed under status bits 12 through 17, as well as any additional device dependent errors in bits 6 through 11. "SR" A 1 indicates uncleared, or unresolved, error conditions. A 0 means no outstanding error conditions.
3 - UNIT AVAILABLE	_(Real time controlled in SR) "ISR" Status interrupt generated, and bit is set, when the device becomes ready for a new data transfer cycle. For example, the data buffer of the device is released from its previous operation. "SR" A 1 indicates unit is now available for the start of the next data transfer operation. A 0 means the device is in full operation and is not available for any new commands.
4 - WORD COUNT REACHED ZERO	_Status interrupt generated, and bit is set, when the Word Count register reaches zero. This bit is cleared by loading a new value into the Word Count register.
5 - DATA CHAINING	_Status interrupt generated, and bit is set, (in place of bit 4) when the Word Count register reaches zero in data chaining mode. This occurs only if bit 16 of the preceding Word Count register load was a 1. Cleared when loading a new value into WC.
6 - 11 DEVICE DEPENDENT	_Errors and condition indicators unique to different devices (see example in section 8.6). Appearance of these bits within ISR as well as SR is also device dependant.

Bits 12 through 16 appear, as described, only in SR. These positions have device dependent functions within ISR.

12 - DMA ADDRESSING ERROR	_DMA reference attempted outside of memory space installed on system. For Virtual main store systems, DMA reference to a non-resident page. Sets the "Error" bit (2). Reset by the "Clear Error" command.
13 - DMA EXCEPTION	_DMA encountered a main store parity error, or other hardware malfunction. Sets the "Error" bit (2). Reset by the "Clear Error" command.
14 - ILLEGAL DATA	_An illegal data character or pattern has been received by the device controller from the device. For example, a card reader may have encountered a non-EBCDIC Hollerith code when operating in EBCDIC mode. Sets the "Error" bit (2). Reset by the "Clear Error" command.
15 - TRANSLATOR ERROR	_Translator malfunction has occurred in a device controller equipped with a data translator. Sets the "Error" bit (2). Reset by the "Clear Error" command.
16 - LOST DATA	_Lost data occured due to latency of the CPU or overloading of the DMA path. Sets the "Error" bit (2). Reset by the "Clear Error" command.
17 - COMMAND REJECT	_Illegal command sent to the device controller. Usually an unrecognized command or a command issued too early or out of sequence. Sets the "Error" bit (2). Reset by the "Clear Error" command. This error indicator appears in SR.

8.4.2.3 INTERRUPT MECHANISM

Interrupts are signals which notify an active CPU process that some change of state has occurred, usually at an I/O device. There are five phases in the interrupt sequence. These phases are called: requested, latched, pending, acknowledged, and released (these names should not be confused with the CPU interrupt terminology described in sections 4.5.2 and 5.8.2). Three flags are involved with this sequence, these are the "request flag" and the "latch flag", located within the device controller, and the "pending flag", which is located within the channel controller. Status and data interrupts must be enabled at the device controller before this sequence can start.

The interrupt becomes "requested" when a device signals its device controller of a condition that has been enabled to trigger an interrupt. An interrupt may also be "requested" when a condition within the device controller itself requires an interrupt. This action is asynchronous and independent of the controller state.

When the channel controller interrogates device controllers for interrupts, it searches for "request flags". Upon encountering one it sets the appropriate interrupt "latch flag" (one exists for each data-in, data-out, and status interrupt). The interrupt is now in the "latched" phase. After the interrupt has been latched, the "pending flag" in the channel controller will be set. The setting of the "pending flag" is immediately followed by a signal to the associated CPU interrupt line (see section 8.3.1, for further discussion).

Since more than one interrupt may be "latched" simultaneously it is necessary for the channel controller to select the one with the highest priority. In the time interval between the "latched" and "pending" states the channel controller has searched the "priority chain" connected device controllers and has selected the highest priority (closest connection to the channel controller) "latched" device. Now the selected interrupt is "acknowledged" by the channel controller. The selected device is given use of the channel until the interrupt is "released" by the CPU I/O process or the DMA controller, whichever is involved with the active interrupt.

The data interrupt and status interrupt mechanisms are fully independent. Any of the phases of the interrupt sequence, of either type of interrupt, can be maintained regardless of the phase of the other.

The "release" of an interrupt is necessary to reactivate the scan for another interrupt of the same type, by both the channel controller and device controller. The "release" is caused by different command sequences, based on the type of interrupt.

Following a:	The interrupt is released:
Data-in interrupt	-after exactly two RIO signals have been transmitted to the channel.
Data-out interrupt	-by RIO followed by a "Data available" channel command (see section 8.3.2).
Status interrupt	-by the first RIO following a "Status Request" channel command (see section 8.3.2).

The device controller recognizes the appropriate sequence, and produces the release signal which clears its "request flag" and "latch flag", and the "pending flag" within the channel controller.

8.4.2.4 DIRECT REQUEST MECHANISM

The direct request mechanism allows an active CPU process to read the content of the device controller registers. A "direct request" may be made at any time except when the channel is in the middle of one of the following operarions.

Data-in interrupt:	between the first and second RIO signal.
Data-out interrupt:	between the RIO and "data available" command.
Status interrupt:	between "Status request" and the next RIO.

"Direct request" is accomplished by executing any of the following device commands:

1) Read Status register.

2) Read Word Count register (I/II).

3) Read Buffer Address Register (I/II).

4) Read Device Control Word A.

5) Read Device Control Word B.

6) Read DIB / Read Odd.

Device ID will be supplied along with each Direct Request. (Refer to section 8.5.3 for complete information on the use of these commands.)

After sending a "Direct Request" the program must wait for the two way delay along the channel cables (300 nanoseconds for 75 foot standard lengths). This means that if an XIO, executing a "direct request", is issued at T-period T(0) the RIO which gates the data returned must not be specified before T-period T(5). Completion of the "direct request" sequence will return the channel to exactly the state which existed prior to "direct request mode". An additional time delay, equivalent to that required for a "direct request" operation, must be allowed by the program following the sequence before allowing any new interrupts to take effect (ie. if there is an interrupt pending, BAR may not be read before 300 ns. after RIO of Direct Request). This is required in order to guarantee the return of any previous information to the channel data and ID lines. Refer to section 8.5.6 for an example of "direct request".

8.4.2.5 DEVICE COMMANDS

Each device controller has a control unit which recognizes and executes "commands" from the CPU, when accompanied by the matching "device select" code assigned to the controller. Device commands may be used to clear a device, start an operation, read device control registers, etc.

The following list specifies standard device commands, as assigned by NANODATA to cover most external device control functions:

CODE (Octal)	COMMAND	DESCRIPTION
01	Clear Device	_Clears (zeroes) all registers, resets all interrupt flags, and disables further interrupts within the device controller. Stops any activity within the devices attached. Also results in the same effect as the "Disable Interrupts" command (05, below).
04	Enable Device Interrupts	_Enables the generation of interrupts, of any of the three types (data-in, data-out, status), by the device controller. If an interrupt is in the "requested" condition, at the time of this command, it will be generated.
05	Disable Device Interrupts	_Inhibits generation of any device interrupts. This does not stop interrupt "requests" from occurring at the device, but does block the "latching" of any interrupts.
07	Read Status	_A Direct Request command (see sections 8.4.2.4 and 8.5.6) to read the Status Register (SR) of the specified device controller on the selected channel.
10	Decrement Word Count	_Decrements Word Count Register if the Allow Decr WC bit is set in DCWB
11	Load Word Count	_Loads the value on the selected channel data lines into the Word Count Register (WC) of the specified device controller.
12	Load Buffer Address Register	_Loads the value on the selected channel data lines into the Buffer Address Register (BAR) of the specified device controller.

21	Read Word Count	_A "direct request" command to read the Word Count register (WC) from the specified device controller on the selected channel.
22	Read Buffer Address Register	_A "direct request" command to read the Buffer Address Register (BAR) from the specified device controller on the selected channel.
24	Read ISR	_A "direct request" command to read the ISR without altering its contents.
25	Read DIB	_A "direct request" command to read the Data In Buffer; undefined results for devices without a DIB.
30	Load Device Control Word A	_Loads the value on the selected channel data lines into Device Control Word "A" (DCWA) of the specified device controller.
31	Load Device Control Word B	_Loads the value on the selected channel data lines into Device Control Word "B" (DCWB) of the specified device controller.
34	Read Device Control Word A	_A "direct request" command to read Device Control Word "A" (DCWA) from the specified device controller on the selected channel.
35	Read Device Control Word B	_A "direct request" command to read Device Control Word "B" (DCWB) from the specified device controller on the selected channel.
41	Start Operation	_Starts the device operation, generating the first data interrupt cycle, if appropriate.
46	Clear Error	_Clears all error bits in the Status Register (SR) and Interrupt Status Register (ISR) of the specified device controller.

8.4.3 DATA ROUTING SUPPORT

8.4.3.1 BUFFER ADDRESS REGISTER (BAR)

The Buffer Address Register serves as a memory pointer for data transfer operations. It is an 18 bit wide register which is loaded by the "Load Buffer Address Register" device command (see section 8.4.2.5). The Buffer Address Register has been incorporated into the device controller in order to remove CPU overhead during data transfers. The content of the BAR will be placed on the channel data lines for use by the CPU, or DMA controller, on the occurrance of any data interrupts. The register may be incremented, or decremented, automatically at the completion of each data word transfer. The CPU program can request this action by enabling the corresponding DCWB bits (see section 8.4.2.1). BAR may also be read at any time, through a "direct request" command (see section 8.4.2.4).

8.4.3.2 WORD COUNT REGISTER (WC)

Word Count is another register designed to minimize CPU overhead during data transfer operations. WC will normally contain the value representing the length of the memory block (buffer) addressed by BAR. It is loaded using the "Load Word Count" device command (see section 8.4.2.5). If enabled by the appropriate DCWB bit (section 8.4.2.1) WC will be decremented, automatically, every time a word transfer has been completed. Upon decrementing to zero the "Word count reached zero" Status Register (SR) bit is set. The equivalent bit in the Interrupt Status Register (ISR) will be set, and a status interrupt generated, if the "Word Count Zero Interrupt" condition is enabled in DCWA (see section 8.4.2.1). Further data interrupts are inhibited once WC reaches zero. Many devices may also be enabled to complete their operating cycle automatically, upon WC reaching this condition. WC may be read at any time by a "direct request" command (section 8.4.2.4).

The Word Count Register is 16 bits wide. It may contain values between 0 and (2**16)-1. The next higher bit (bit position 16) is used to indicate a "data chaining" mode of operation, on the current data block (see section 8.4.3.4).

An I/O device may be operated without using the Word Count register by simply leaving the "decrement word count" control (DCWB bit 6) reset (0). The Word Count register itself must be loaded with a non-zero value, if data interrupts are to be allowed to occur. The programmer must be aware that this becomes a hazardous mode of operation, as there is now no hardware protection against a data transfer overrunning its memory buffer boundary.

8.4.3.3 DATA TRANSFERS

Data transfers are maintained on a word by word basis. All of the units described so far, in section 8.4, establish a system which allows data transfers to be performed with little CPU overhead. At the same time that a data interrupt signal is being sent to QM-1, the Buffer Address Register and Task ID are being presented to the I/O channel. The program responding to the interrupt need not know which device actually generated the interrupt.

The nanostore location to which the data interrupt is associated determines the operation to be performed for that interrupt. Separate nanostore locations are required for each data-in and data-out interrupt possible. The data routing procedure applied may simply ignore the Task ID and route the data word to or from the main store location identified by the content of the BAR presented to the channel. The procedure may optionally use the Task ID to determine to which of the QM-1 memories the current data word should be routed (such as main store, control store, etc.).

On data-out interrupts, where data is to be routed from the CPU to the device, the program immediately issues RIO thereby reading BAR into the port register. This RIO also switches the device controller to a "waiting for data" condition. When the "data available" channel command (section 8.3.2) is sent, the device controller accepts the data then present on the data lines. There is no need for any time delay between the RIO that reads BAR and the "data available" command.

In the case of data-in interrupts, the channel controller expects two consecutive RIO signals to complete the transfer. The first RIO gates BAR into the port register and signals the device controller to place the data word onto the channel data lines. The channel needs about 300 nanoseconds (for a 75 foot cable) to respond to the first RIO before the data is available for the second RIO. This means that if the first RIO is sent at T-period T(0) the second cannot be issued before T-period T(5).

8.4.3.4 DATA CHAINING

Data chaining is a mechanism that assists the I/O process in combining separate blocks of data in memory into one logically consecutive buffer space. In order to invoke this mode of operation bit 16 of the Word Count Register (section 8.4.3.2) must be set to a one. This is accomplished at the same time the actual word count value, describing a data segment length, is loaded into the lower 16 bit positions of WC using the "Load Word Count" device command (section 8.4.2.5). With the proper interrupt enables allowed, when WC reaches zero it will generate a status interrupt and will set the "data chaining" bit in the Status Register (SR) and Interrupt Status Register (ISR). The "data chaining" status bit replaces the normal setting of the "word count reached zero" bit in the ISR.

Following a "data chaining" status interrupt the I/O control process must fetch both the BAR and WC information that identifies the next buffer segment. This new data routing information is transferred to the device controller using the "Load Buffer Address Register" and "Load Word Count" commands. As soon as this is accomplished a "start operation" command (section 8.4.2.5) is issued to activate the continuation of the data transfer, using the new BAR and WC values. The transfer of the last buffer segment, of a set, is indicated by the presence of a "zero" data chaining mode bit in the WC register. See section 8.5.7 for additional discussion.

8.4.3.5 DIRECT MEMORY ACCESS (DMA)

Direct Memory Access (DMA) mode of operation permits direct data transfer between I/O devices and QM-1 main store. No CPU intervention is required for the duration of direct memory transfers. This reduces data handling overhead for the CPU, and also permits the attachment to the system of very high speed I/O devices that would otherwise be too fast for normal CPU data interrupt response. DMA also permits highly efficient nanocoding of non-interruptable, long duration, computational processes; since the CPU is not required to handle frequent data interrupts.

Any standard device controller may be switched to DMA mode, as long as the channel is equipped with the DMA option and a multi-port main store system. To utilize a DMA data path, device operation is initiated in the same manner as for regular CPU data transfers. All device controller registers should be loaded to initialize the transfer as if it were to be via a CPU port. The only difference is that bit 9 of DCWB must be set to "one" (DMA mode bit, see section 8.4.2.1). When the device and controller have been prepared for the data transfer the program should issue the appropriate "start operation" device command (see section 8.4.2.5). From this point on, the device controller communicates directly with the memory access controller.

Switching a device to DMA mode does not disconnect it from the QM-1 channel. All of the facilities to "measure the pulse" of, or to override, the current I/O operation remain active. For example, "direct request mode" may be applied at any time to sample an active register such as BAR or WC. Any enabled status interrupt may occur as usual. The normal termination of the DMA data block transfer will usually occur when WC reaches zero. The CPU may be asynchronously notified of this event through the "Word count reached zero" status interrupt, as indicated by the corresponding bit being set in the Interrupt Status Register. Abnormal termination will occur as a result of any preselected error interrupt condition. Two "error" status indicators are associated with DMA mode (also refer to section 8.4.2.2).

DMA ADDRESSING ERROR - indicates an attempted reference beyond the memory space installed on the computer system. For those systems equipped with the Virtual Main Store option this bit indicates that a reference to a non-resident user page was attempted.

DMA EXCEPTION - indicates that DMA encountered a main store parity error, or some other hardware malfunction.

8.5 I/O SYSTEM OPERATIONS

8.5.1 GENERAL

This section discusses actual I/O System operations and procedures. Several actual nanocode examples are included as an illustration of the various coding techniques applicable in I/O programming.

8.5.2 INITIALIZATION

Channel initialization is usually performed only when the state of either the channel or its devices is unknown. This normally will be required following a system power up, or after a channel has been deactivated for maintainance activity. Initialization will be performed by Operating Systems during system initial program loading and by stand alone processes at the start of execution.

This section describes several aspects of initializing a channel or a device. It is more suggestive than imperitive since this operation can be performed in several ways.

The first step of initialization is to clear the channels and devices. Three means of generating the clear signal are available:

1. MASTER CLEAR (see section 8.2.2) clears all channels and devices along with the entire CPU.

2. I/O RESET (see section 8.3.2) clears the channel and all devices on the "KA" selected port.

3. CLR COMMAND (see section 8.4.2.5) will clear the device controller to which it is sent.

Two device commands "enable all device interrupts" and "disable all device interrupts" (see section 8.4.2.5) act as "connect device to the channel" and "disconnect device from the channel" respectively. Following a channel or device clearing operation device interrupts are left disabled. Disabling the device interrupts removes the device from the priority chain and does not allow any device originated activity. However it is still possible to load any register of the device controller as well as to read any register and to check status, through direct request commands (see section 8.5.6).

It is recommended that each device control program clear the device controller and load its initial control registers before issuing "enable device interrupts" and "start operation" commands.

8.5.3 COMMAND EXECUTION

Commands (channel and device, see sections 8.3.2 and 8.4.2.5) are generated by setting the device number on the "device select" lines and the command code on the "command" lines, and then executing XIO. The following are two nanoprogram segments to generate "I/O RESET" (channel command) and to "LOAD DCWA" (device command).

```
* SEND I/O RESET (70) TO CHANNEL 2.
: "SINCE I/O RESET IS A CHANNEL COMMAND NO DEVICE SELECTION IS NECESSARY"
....    KA = 2 "PORT",  KB = 70 "CODE FOR I/O RESET FUNCTION"
X...    KB->FIPH "PLACE CODE ON COMMAND LINES"
        XIO      "TRANSMIT COMMAND"
        READ NS  "FETCH NEXT NANOWORD"
.X..    GATE NS  "PROCEED TO NEXT NANOPROGRAM SEGMENT"

* LOAD DCWA OF DEVICE 17 ON CHANNEL 4.
: "ASSUME DATA TO BE LOADED IS ALREADY IN PORT REGISTER 4"
....    KA = 4 "PORT",  KB = 30 "CODE FOR LOAD DCWA",  KX = 17 "DEVICE ID"
X...    KB->FIPH "PLACE CODE ON COMMAND LINES"
        G(G KX)  "PLACE DEVICE ID ON DEVICE SELECT LINES (G-BUS)"
        XIO      "TRANSMIT COMMAND AND DATA"
        READ NS  "FETCH NEXT NANOWORD"
.X..    GATE NS  "PROCEED TO NEXT NANOPROGRAM SEGMENT"
```

8.5.4 DATA TRANSFERS

Data transfers are normally invoked by a data interrupt. On any data interrupt the channel will present the contents of BAR (Buffer address register) for gating into the port. The issuing of an RIO on "data in" interrupts will gate the BAR into the port register, switch the device to data mode, and then cause the device to put its data on the data lines for later gating into the port. The next (second) RIO then gates the data into the port register and generates the release signal to the channel (see timing note below). In the case of "data out" interrupts a "data available" channel command replaces the second RIO for releasing the channel. Updating of BAR and WC (word count) occur, if enabled, on the second RIO for the "in" case and on "data available" for the "out" case.

TIMING NOTE: The following restriction must be observed. The RIO which gates the data (the second RIO after a data-in interrupt) may not be issued before the 5th T-Period after the first RIO.

SAMPLE DATA OUT INTERRUPT HANDLER

The following code will read the BAR of the interrupting device, use that value to address one word of main store, read that MS location and send the data received to the interrupting device; thus releasing the channel and satisfying the data-out interrupt.

```
*****************************************************************************
*                        EQUATES FOR THIS FUNCTION                          *
*****************************************************************************

        PORT NUM  = 4                "PORT 4 IDENTITY"
        PORT PATH = PORT NUM + 32.   "VALUE 36. FOR ADDRESSING ES(4) AS LS(36)"
        DATAVAIL  = 71               "VALUE OF DATA AVAILABLE CHANNEL COMMAND"

*****************************************************************************
*  DATA OUT INTERRUPT HANDLER:                                              *
*  ROUTES ONE DATA WORD FROM MAIN STORE TO CHANNEL 4                        *
*  RETURNS CONTROL TO THE INTERRUPTED PROCESS FOLLOWING TRANSFER            *
*****************************************************************************

:       "DATA OUT INTERRUPT LEVEL XX, ENTRY POINT"
....    BRANCH(N.+1), KA = PORT NUM, KB = PORT PATH, KX = MS BUSY
S...    GATE NS(NOT X)        "AWAIT MAIN STORE NOT BUSY"
        KB->FMIX, KB->FMOD    "SET MAIN STORE PATH TO EXTERNAL STORE"
.X..    RIO                   "GATE BAR TO PORT REGISTER, ES(4)"
        READ NS               "PREPARE TO CONTINUE TO NEXT NANOWORD"
..X.    READ MS               "FETCH NECESSARY MAIN STORE WORD"
        GATE NS               "CONTINUE IN NEXT NANOWORD"

:       "COMPLETION OF DATA OUT INTERRUPT LEVEL XX"
....    ALLOW INTS "ON EXIT", KX = MS DATA, KB = DATAVAIL, KA = PORT NUM
S...    GATE NS(NOT X)        "AWAIT MAIN STORE DATA AVAILABLE"
        GATE MS               "GATE MS DATA EACH TIME UNTIL SUCCESSFUL"
.S..    XIO                   "SIGNAL CHANNEL CONTROLLER OF INTENTIONS"
        KB->FIPH              "DATA AVAILABLE COMMAND"
        READ NS               "READ NEXT NANOWORD OF INTERRUPTED PROGRAM"
..X.    GATE NS               "CONTINUE INTERRUPTED PROGRAM"
```

8.5.5 STATUS INTERRUPT HANDLING

The status interrupt mechanism of the device controller is described in section 8.4.2.2. When the CPU senses a status interrupt it may issue a "status request" channel command. The interrupt status will be available for gating into the port after the 5th T-Period following the status request. A status request may not be issued in the middle of data interrupt handling, ie. after the RIO that read BAR into the port.

The following nanoword will read the interrupt status of the interrupting device on channel 4 into LS(G.STAT). Where G.STAT represents a G-register that points to the local store register to receive the status. G.DEV is used to represent another G-register which will itself receive the Device ID returned by the interrupting device.

```
*  THE STATUS INTERRUPT HANDLER WILL TRANSFER CONTROL TO THE SYSTEM SUPERVISOR
*  TO ANALYSE THE STATUS RETURN, AND TO CHANGE THE OPERATING STATE IF NEEDED.
:       "STATUS INTERRUPT LEVEL YY, ENTRY POINT"
....    BRANCH(SYSTEM INT) "SUPERVISOR TRANSFER ADDRESS"
        KA = 4 "PORT",  KB = 77 "STATUS REQUEST COMMAND"
S...    XIO, KB->FIPH      "PLACE CHANNEL IN STATUS MODE"
.S..    G(G.STAT)          "IDENTIFY G REGISTER"
        G->FEOD, KA->FEOA "SET UP EXTERNAL STORE TO LOCAL STORE PATH"
..X.    IO ID->G.DEV       "SAVE THE DEVICE ID, WHICH IS NOW AT THE PORT"
...S    RIO, GATE ES       "READ THE PORT AND PASS THE STATUS TO LS(G.STAT)"
        READ NS, GATE NS  "TRANSFER CONTROL TO THE SYSTEM SUPERVISOR"
```

8.5.6 DIRECT REQUEST COMMANDS

The direct request commands allow a QM-1 program to examine all of the control registers of a device without affecting the current state of the channel or device. After sending a direct request command the CPU program must wait for a minimum of 5 T-Periods before gating any expected data into the port. The program must then send an RIO to read the desired data into the QM-1 port register. The RIO also returns the channel to whatever state it was in prior to the direct request. For those direct request commands that output data no time delay is required. In this case the direct request transmits the command and any data immediately. An RIO is not allowed, and is not expected by the channel.

An additional timing note: Data interrupts occur at the moment when the BAR data is immediately available at the port data lines, and the "task ID" is available on the device ID lines. This information is expected by the data interrupt handlers. Therefore, when a direct request has temporarily changed the information content on those lines (ie. an "input word count register" command) the process in control must guarantee at least another 5 T-Periods (300 nanoseconds), after release of direct request mode, to allow return of the original data-mode information.

A simple nanoprogram to read the DCWA register might be as follows.

```
RDCW:   "DCWA OF DEVICE B ON PORT 4 IS READ INTO THE PORT REGISTER"
....    KA = 4 "PORT", KB = 34 "COMMAND TO READ DCWA"
S...    XIO, G(G B), KB->FIPH  "SEND DIRECT REQUEST TO DEVICE B"
.S..
..X.    READ NS                "WAIT AT LEAST 5 T-PERIODS"
...X    RIO, GATE NS           "READ INTO PORT REGISTER AND EXIT"
```

8.5.7 DATA CHAINING

Data Chaining is a facility which enables the program to connect separate blocks of data in main memory as one "consecutive" buffer. The device controller contains basic logic to support this facility. Several optional methods exist, each providing varying degrees of efficiency for effecting data chaining, though only the most elementary QM-1 method is described.

To permit data chaining the supporting program must provide a list of buffer addresses and data lengths in advance of starting the chained operation. Initially the first buffer segment starting address is loaded into the BAR, and the word count plus data chaining indicator (bit 16 of the word count value) is loaded into WC. The device controller will sence WC going zero and generate a status interrupt. In this case the interrupt status register will be found to have bit 5 (data chaining in effect) set in place of the usual bit 4 (word count going zero). The status interrupt handler must then fetch the next set of words identifying the following buffer segment, from the buffer list, and transfer them to the BAR and WC. This is followed by a "start" command which continues the original data transfer operation with a data interrupt.

Data chaining procedures for devices with high transfer rates require more automated methods of chaining. These methods are unique to each high speed device, and are discussed indepenently within their respective device controller manuals.

8.6 EXAMPLE OF AN ACTUAL DEVICE CONTROLLER

8.6.1 DEVICE SPECIFICATIONS

This section describes the device controller and operation of the NANODATA LP135 line printer. The LP135 is a low speed device capable of outputting hard copy comprised of printed lines, up to 132 characters wide, at a rate of 135 lines per minute. There are 64 character codes available, corresponding to USACII codes 40 to 137 (octal). The LP135 also has a four channel vertical format tape.

The printer has a full 132 character line buffer. To print a line the characters are first loaded into the line buffer. Then, a print signal is issued causing the line to be printed and the forms advanced to the next line. While the forms are advancing the line buffer may be re-loaded, in order to be ready for the next print cycle when the forms are in position. If less than 132 characters have been sent to the buffer, when the print signal is issued, the buffer is automatically filled out with blanks before the actual print operation takes place.

The LP135 also has provision for form feed controls, and four format channel indicators. The term "skew" is used, in this discussion, to indicate forms motion through the printer. "Skew mode" represents the continuous forms motion through the printer, beyond one integral line. The next sections describe the control of the LP135 in conjunction with the standard device controller on the QM-1.

8.6.2 CONTROLLER SPECIFICATIONS

The following lists are descriptions of the printer status registers, device control words, and the printer device commands. The lists specify standard as well as device dependent functions.

8.6.2.1 DEVICE STATUS

STATUS REGISTER USAGE (SR AND ISR)

Refer to section 8.4.2.2 for a general discussion of the status registers.

Bit position & ident		Description of status	Appears in ISR/SR
0	Device Ready	_A "1" indicates that power is applied to the print unit, paper is properly installed, and the ready button has been depressed.	BOTH
1	Device Not Busy	_A "1" indicates the device is motionless, and is not in operation on any computer issued commands.	BOTH
2	Error Condition	_The logical "OR" of status register bits 12, 13, 16, and 17.	BOTH
3	Buffer Ready	_The printer's line buffer is ready for character loading (see section 8.4.2.2).	BOTH
4	Word Count Zero	_(described in section 8.4.2.2).	BOTH
5	Data Chaining	_(described in section 8.4.2.2).	BOTH
6	Line Buffer Full	_The 132 character line buffer is full. A "real time" indication of line buffer state. This condition is prerequisite to an actual physical print cycle, but is automatically produced following a "print" command to the device controller.	BOTH
7	Line Count	_A status interrupt produced, only if enabled, every time the printer passes the start of a line position during forms motion. If the printer is in "slew mode" the program has one millisecond to stop the "slew" in order to stop on the current line. This bit is reset by the "clear error" command (see section 8.4.2.5).	ISR
8	Format Channel 1	_A status interrupt produced, only if enabled.	SR

		SR bit indicates a Format Channel 1 punch was encountered.	
9	Format Channel 2	_Same as above, for channel 2.	SR
10	Format Channel 3	_Same as above, for channel 3.	SR
11	Format Channel 4	_Same as above, for channel 4.	SR
12	DMA Addressing Error	_Standard (see section 8.4.2.2)	SR
13	DMA Exception	_Standard (see section 8.4.2.2)	SR
14	Format Channel	_The "OR" of the SR(8,9,10,11) bits. A Format Channel interrupt is produced if enabled, when a Format Channel is encountered on the forms control tape, during carriage motion. If in "Slew" mode, the program has one millisecond to stop the Slew on the desired line. The bit is cleared when the next line is encountered.	ISR
16	Lost Data	_A status interrupt produced, only if enabled, when an attempt has been made to load the line buffer when the buffer is not yet ready for a new character. This device does not demand that characters be transferred to its line buffer within a limitted time frame. Therefore, this error usually will indicate a hardware malfunction.	SR
17	Command Reject	_A status interrupt is generated if "command reject" is enabled, when any of the following conditions occur: 1) Any form feed command issued when the unit is not available (see status bit 3, above) or is not ready (status bit 0). 2) An attempt to clear the line buffer while a print cycle is in progress. 3) An attempt to print a line without one of the following status conditions being true: A) Buffer Ready (Unit Available). or B) Buffer Full.	BOTH

8.6.2.2 DEVICE COMMANDS

PRINTER DEVICE COMMANDS

Refer to section 8.4.2.5 for a complete discussion on device commands. All command code values are shown in octal.

CODE	COMMAND	DESCRIPTION
01	Clear Device	_Standard (see section 8.4.2.5).
04	Enable Device Interrupts	_Standard.
05	Disable Device Interrupts	_Standard.
07	Read Status	_Standard.
11	Load Word Count	_Standard.
12	Load Buffer Address Register	_Standard.
21	Read Word Count	_Standard.
22	Read Buffer Address Register	_Standard.
30	Load Device Control Word A	_Standard.
31	Load Device Control Word B	_Standard.
34	Read Device Control Word A	_Standard.
35	Read Device Control Word B	_Standard.
40	Start Operation and Print	_This command combines the functions of commands 41 and 42, described below. It causes automatic physical line printing to occur, as would be effected by command 42, immediately following the Word Count register reaching zero. In general, this command initiates one full line printer operating cycle; from data transfer to completion of physical line printing, and automatic advance to the next. Refer to commands 41 and 42 for additional detail.
41	Start Operation	_Assuming the printer is ready, and the device controller registers are properly initialized, with WC containing a non-zero value, a data block transfer

		will be started. "Start Operation" triggers the first data-out interrupt request to the CPU or DMA (depending on DMA mode setting), for the current data block.
42	Print	_The printer will print the current contents of the line buffer and then automatically advance to the next line. If the buffer is not full it will be automatically blank filled. This command may not be issued unless either the "buffer ready" or "buffer full" condition is true (see status above).
45	Clear Buffer	_Blank fills the entire printer line buffer, and sets the buffer to an empty condition. May be issued only if the "buffer ready" status bit is true.
46	Clear Error	_Standard (see section 8.4.2.5).

Note: All of the forms motion (control) commands described below require the "buffer ready" status condition.

50	Space (Line Feed)	_Causes the printer to advance the forms immediately to the next line.
51	Skip To Format Channel 1	_Start forms "slew". Stop when format channel 1 is encountered.
52	Skip To Format Channel 2	_Same as 51, for channel 2.
53	Skip To Format Channel 3	_Same as 51, for channel 3.
54	Skip To Format Channel 4	_Same as 51, for channel 4.
64	Start Slew	_Starts "slew mode" continuous forms motion. Motion continues until a "Stop Slew" (65), "Clear Device" (01), or "I/O Reset" (70) command is issued.
65	Stop Slew	_Stops forms motion, previously issued by any of the forms control commands (above).

8.6.2.3 DEVICE CONTROL WORDS

PRINTER DEVICE CONTROL WORDS (DCWA and DCWB)

Refer to section 8.4.2.1 for a discussion of standard DCW functions.

Bit Identity		Description
	DATA MAPPING CONTROLS	
DCWA 02	TRANSLATE	_If set, the printer controller will translate lower case ASCII characters to upper case.
DCWA 03	PACK/UNPACK	_If set, the printer controller will unpack two 8-bit bytes from each 18 bit word transferred. The leftmost character (bit positions 8 to 15) is accessed first. The characters within the 18 bit word are right justified. The high order two bits are ignored. If the Word Count register is being employed during "packed" mode operation it will be decremented by two for each QM-I data word transferred, thereby becoming a character count.
	STANDARD STATUS INTERRUPT ENABLE MASKS	
DCWA 04	COMMAND REJECT	_Standard (see section 8.4.2.1).
DCWA 05	ANY ERROR CONDITION	_Standard.
DCWA 06	DEVICE READY	_Enables a status interrupt whenever the printer ready or reset buttons cause a change in the printer's ready condition.
DCWA 07	NOT BUSY	_Enables a status interrupt whenever the printer completes a full operating cycle. Indicates an end of "slew mode" operation as well as an actual print cycle.
DCWA 08	UNIT AVAILABLE (Buffer Ready)	_Enables a status interrupt when the line buffer becomes ready for character loading.
DCWA 09	WORD COUNT ZERO	_Standard.

PRINTER DEPENDENT STATUS INTERRUPT ENABLE MASKS

DCWA 10	FORMAT CHANNEL 1	_Enables a status interrupt each time Format channel 1 is encountered.
DCWA 11	FORMAT CHANNEL 2	_Same as DCWA 10, for channel 2.
DCWA 12	FORMAT CHANNEL 3	_Same as DCWA 10, for channel 3.
DCWA 13	FORMAT CHANNEL 4	_Same as DCWA 10, for channel 4.
DCWA 14	LINE COUNT	_Enables a status interrupt every time a new line position is passed, during forms motion.
DCWA 15	BUFFER FULL	_Enables a status interrupt when the printer line buffer becomes full.

All DCWB function for this device are standard. Refer to section 8.4.2.1 for the discussion of the DCWB controls.

8.6.3 OPERATION AND PROGRAMMING

The following briefly describes basic printer operations. Printer operations can be grouped into four categories: Initialization, Status, Data transfer and print, and Forms control. Due to the general flexibility of the device controller, certain conventions must be set up to govern the specific forms of operation desired. The rules listed below are enforced only for this example, and by some of the characteristics expected from the device described.

1) Only upper case ASCII characters will be transmitted to the printer.

2) One character is accessed per 18 bit word, and is right justified.

3) One line, or fraction of a line, is considered to be a "data block".

4) Data routing support units, BAR and WC, will be incorporated in data transfers, and WC will be used to terminate the "data block" transfer.

5) Data transfers will be from main store memory only.

6) Data chaining will not be incorporated in the data transfers described.

8.6.3.1 DATA TRANSFER OPERATION

INITIALIZATION:

The general initialization procedure is described extensively in section 8.5.2. The following is specific to this print operation. Assume that the printer is known to be ready, and operational. The DCW Registers are loaded.

The following DCWA status interrupt enable masks are set:

DCWA 05	Any Error Condition
DCWA 06	Device Ready
DCWA 07	Not Busy
DCWA 08	Buffer Ready
DCWA 09	Word Count Zero

The following DCWB data routing support controls are set:

DCWB 06	Decrement Word Count
DCWB 07	Increment Buffer Address Register

All other DCW bits are left reset (zero). BAR is then loaded with the first word address of the data block in main store. WC is loaded with the number of words (characters) for the line to be printed. The printer is now ready for the start of its first operation, which is described under data transfer below.

STATUS:

The device status can be read at any time via the "direct request" mechanism (see section 8.4.2.4). A status interrupt may occur on any of the conditions enabled by DCWA and DCWB. Status information accessing operations are standard, and are extensively described in section 8.4.2.2.

DATA TRANSFERS:

After the printer controller has been initialized, as described above, the status should be read to make sure that no abnormal condition exists and that the printer is ready for the transfer: device ready, not busy, buffer ready. The program must issue an "enable device interrupts" (04) command which links the device into the channel priority chain, and allows interrupts to be transmitted to the CPU.

The "start operation" (41) command will generate the first data-out interrupt. Data transfers are further described in sections 8.4.3.3 and 8.5.4. After each word (character) has been received by the printer controller, and transferred to the line buffer, the controller will update the values in BAR and WC and then will generate the next data-out interrupt. When WC reaches zero a status interrupt is generated and the corresponding SR and ISR bits are set.

Before the physical print cycle is initiated, the program should check whether the printer is ready for the "print" (42) command (see the command list in section 8.6.2.2, above). The "print" command is then issued. The end of the print cycle will result in both the "buffer ready" and "not busy" status indications occurring together (unique to the LP135 printer). Following this status interrupt the printer controller may be prepared for its next operation.

8.6.3.2 DEVICE CONTROL FUNCTIONS

All forms control commands may be issued only when the printer "buffer ready" status is set ("1"). Data transfers to the line buffer may be performed even while a form feed, or other "slew mode", operation is in progress, in preparation for an immediately following print cycle.

A "space" (50) command will advance the printer exactly one line position.

A "Skip to format channel 1" (51) command will advance the forms until format channel 1 is encountered. Similarly, commands for format channels 2, 3, and 4 will have the same affect.

Another form feed operation is "slew mode". It is initiated by a "start slew" (64) command. The program may control the amount of forms advance by referencing either the format channels or the line count indicators (see section 8.6.2.1, above). The "slew" is limited to four conventional pages worth of forms at a time, to avoid accidental waste of paper. An automatic "stop slew" is issued by the controller on a runnaway forms condition. To stop the "slew" the program must issue a "stop slew" (65) command, or some form of device clearing command. To stop motion at the line position most recently indicated the "stop slew" command must be issued within one millisecond of notification.

9 INSTALLATION PLANNING ***PRELIMINARY INFORMATION FOR GUIDELINE USE ONLY***

9.1 GENERAL

This section has been prepared to serve as guide for site planning and site preparation for the QM-1 Computing System. It contains the general information for determining floor space, air conditioning and power requirements.

9.2 SPACE REQUIREMENTS

In the standard configuration, the QM-1 CPU consists of 3 bays. The 3 bays are normally assembled into a "Y" configuration as shown below:

```
    /\             /\
   /   \         /   \
  /      \     /      \
 /         \ /         \
/  BAY 3   / \   BAY 2  \
 \        /   \        /
   \     /     \     /
     \  /       \  /
       \---------/
        I       I
        I       I
        I BAY 1 I
        I       I
        ---------
          FRONT
```

BAY 1 is 23.50" wide, 27" deep and 61.75" high
BAY 2 and BAY 3 are both:
27.75" wide, 27" deep and 61.75" high

BAY 2 houses the central processor, control store, and nanostore.
BAY 3 houses main store and the power supplies.

BAY 1 is optional, and normally houses disk drives, tape drives, and controllers.

Access to the internal components of the system is through swing out doors on each side of BAY 2 and BAY 3, and through the front of BAY 1.

BAY 1 may contain one disk drive and controller (NANODATA DD50), and two 12.5 IPS tape drives and controller (NANODATA TT12.5). Alternatively, BAY 1 may contain two disk drives and controller. In either case, a pullout rack extends 23" for service. The remaining service requirements are handled by removing the side panels.

The complete system, with swing out doors fully opened and the rack fully extended, measures 124" wide and 112" deep.

9.3 ENVIRONMENTAL REQUIREMENTS

TEMPERATURE

The ambient temperature of the installation site should be maintained between 60 degrees F and 80 degrees F. The recommended temperatiure is 70 degrees F. Operation within the limits given will have no adverse effect on system system performance.

HUMIDITY

A relative humidity of 40 - 60 percent at the installation site is recommended. Humidity conditions must not allow condensation to occur on any surface or component within the system. Excessively low humidity (ie less than 25% RH) may cause problems with printer paper and cards, due to static charges.

AIR CONDITIONING REQUIREMENTS

The requirements for air conditioning will vary greatly with system configuration, use, and local conditions. A typical installation will produce a thermal load of 20,000 to 25,000 BTU/HR.

9.4 POWER REQUIREMENTS

Nominal power requirements of the QM-1 Computing System are:
    120/208 volts, 3 phase,
    60 Hz @ 7 KVA.
The system is also available (on special order) to operate on 50 Hz.

Unless other power cabling is requested, NANODATA will supply the system with a power cable terminated in a male plug as follows:
    3 pole,4 wire grounding,125/250 volts.
    Hubbell Plug 14-30P, Model Number 9431 or 9432
The power cable may enter BAY 3 at either the bottom or top.

The customer's site should provide the following power:
    One line, 3 phase, 120/208 volts, 4 wire @ 40 Amps, with
    recepticle - Hubbell 14-30R, Model Number 9430 or equivalent.
    Circuit Breaker Panel with
        1 circuit--3 phase/30 Amps
        9 single phase circuits (10 Amp Circuit Breakers - 3 per phase

9.5 PERIPHERAL REQUIREMENTS

The space and power requirements for the peripherals housed in the computer main frame are included with those for the CPU. A large variety of stand-alone peripherals may be attached to the QM-1. The space and power required for certain typical peripherals is listed below.

Line Printer - NANODATA P300 (including paper stand and stacker) 38" deep x 30" wide x 41" high	120 VAC -	AMPS
Card Reader - NANODATA CR200 (top mounted on storage cabinet) 23" deep x 18.5" wide x 43" high	120 VAC -	AMPS
CRT - NANODATA DT2 (mounted on specially designed table) 30" deep x 48" wide x 43.5" high	120 VAC -	AMPS
Tape Drive - NANODATA TT45 (individual cabinets not part of CPU) 21"deep x 26" wide x 58" high	120 VAC -	AMPS

Contact NANODATA Marketing Department for requirements of peripherals not shown here.

9.6 INSTALLATION LAYOUT -- MEDIUM SCALE CONFIGURATION

```
 <------------------------20'------------------------------------->
                                 --------------                          ^
       /\       /\               I            I                          I
      /  \     /  \              I   LINE     I                          I
     /    \   /    \             I   PRINTER  I         ----------       I
    /      \-/      \            I            I         I       I        I
    \ BAY 3   BAY 2 /            I            I         I       I        I
     \             /             ---------------        I TABLE I        I
      \           /                                     I       I
       \         /                                      I       I        12'
        I       I                              ---------I-------I
        I       I                              I        I CRT   I        I
        I BAY 1 I                              I CARD   I   +   I        I
        I       I                              I READER I  TABLEI        I
        I       I                              I        I       I        I
        ---------                              ------------------        I
                                                                         I
                                                                         I
                                                                         V
```

Customers should plan installation details with NANODATA service personnel at least 30 days prior to delivery. This will assure proper cabling, circuit protection, grounding and noise control.

APPENDIX A - QM-1 PORT INTERFACE SPECIFICATIONS

A-1 GENERAL SPECIFICATIONS

This section gives electrical specifications for the user desiring to interface his equipment directly to the QM-1 port. Because of the unique architecture of the QM-1, it is difficult to design external hardware without understanding some of the machine concepts. The hardware designer is particularly encouraged to be familiar with the following sections of the HARDWARE LEVEL USER'S MANUAL:

4.2.5	EXTERNAL STORE
4.3	SIX-BIT CONTROL STRUCTURE
4.5.2.4	EXTERNAL INTERRUPTS
4.6	EXTERNAL INTERFACE
5.3	FUNDAMENTAL TIMING CONSIDERATIONS
5.5	DATA TRANSFER FUNCTIONS
8.1	QM-1 I/O SYSTEM, GENERAL
8.2	QM-1 I/O CONTROLS
8.3	STANDARD CHANNEL CONTROLLER

The port signals are available through the eight port connectors located at the CPU. Interface cards that mate with the CPU port connectors are available from NANODATA. These cards provide for termination of the user cabling to the QM-1, and also have space for some logic, cable drivers, receivers, etc. It is conceivable that a simple device might have all of its interface logic mounted on the interface card. This limits the channel to one such device. The interface card is described in detail in section A-3. Section A-2 gives a full and detailed description of the port signals.

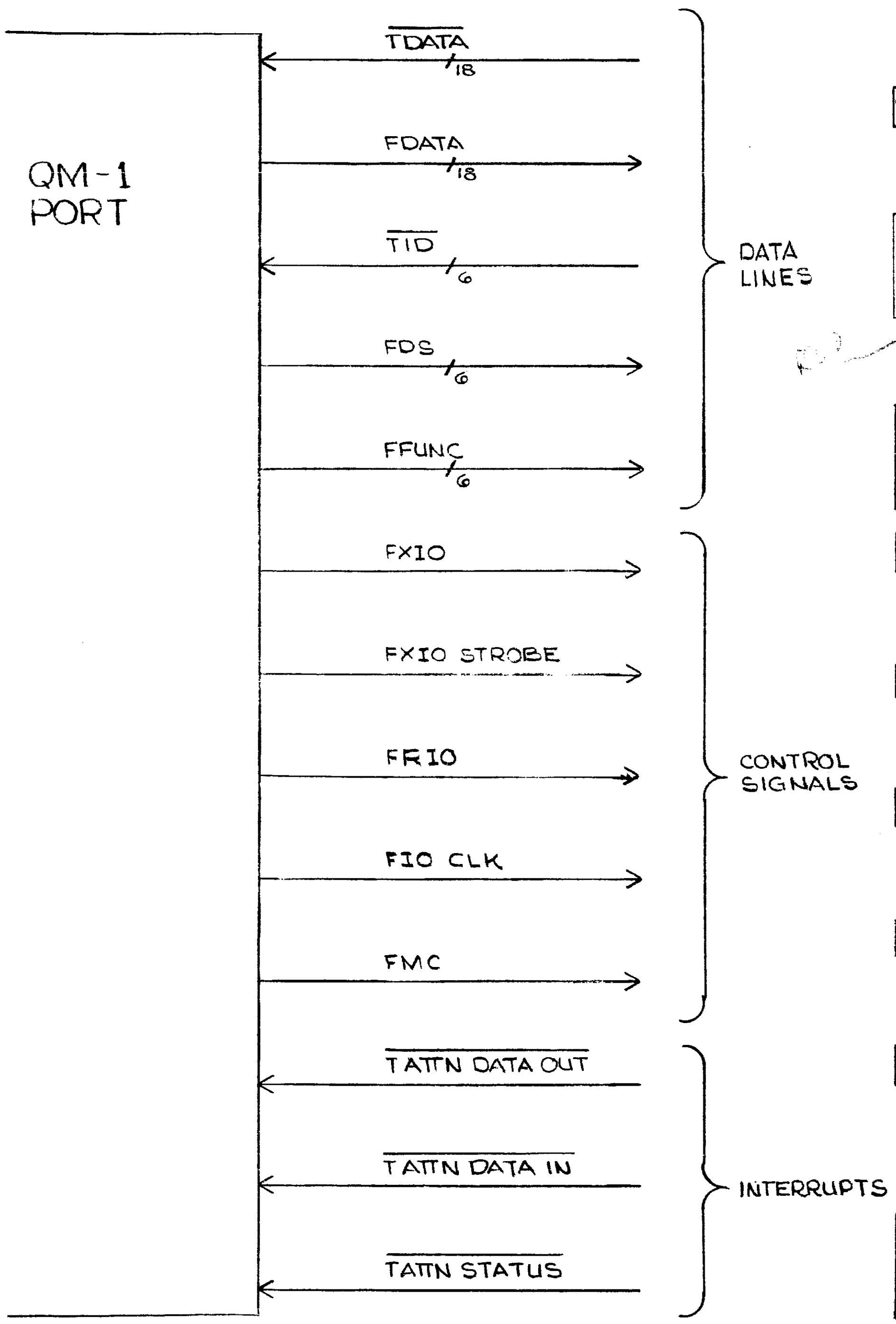


FIG 1 QM-1 PORT SIGNALS

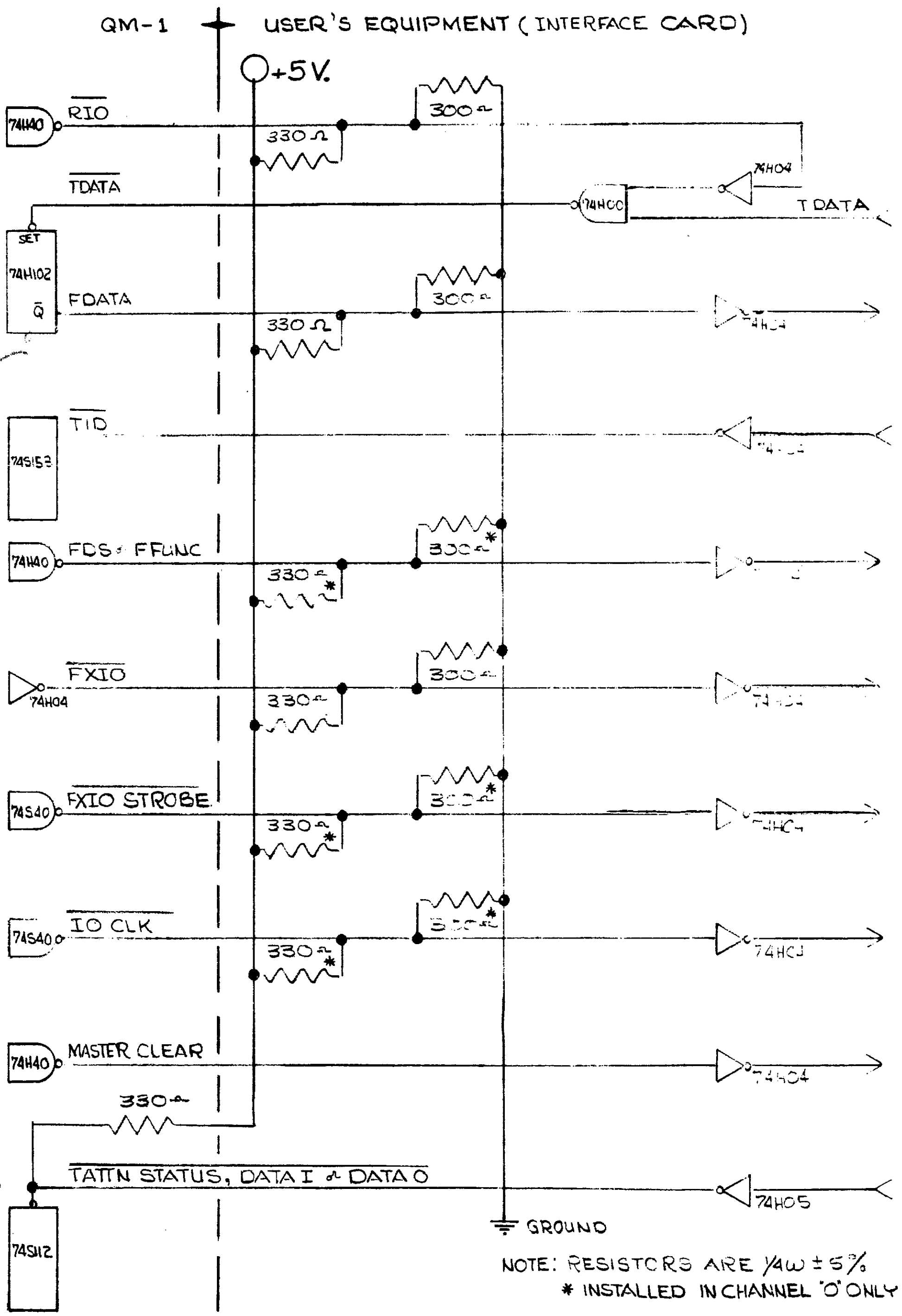


FIG 2 TERMINATION of PORT SIGNALS

A-2 THE PORT SIGNALS

A-2.1 GENERAL

Section A-2 is a detailed description of the functional, electrical and timing characteristics of the port signals. The port signals are shown in figure 1.

A-2.2 FUNCTIONAL CHARACTERISTICS OF THE PORT SIGNALS

The following table describes the functional characteristics of the port signals.

NOTES:

1) The logical level is the relative voltage when the signal is "true". A "low" signal, for example, is a signal whose relative voltage is "low" when it becomes "true". (See section A-2.3 for further discussion.)

2) Caution should be taken with the definitions of "pulse" and "level". Because of the unique architecture of QM-1 the program has full control over the port signals. A "pulse" is defined as a signal that is true for a relatively short time. Programming cannot affect the width or appearance of a "pulse". A signal that is defined as a "level" can be made to look like a "pulse", by allowing that level to remain constant for only one T-period.

3) The port signals were determined with the NANODATA standard I/O scheme in mind. The term "normally used", that occurs frequently in the table below, refers to its use in NANODATA standard I/O channels. These designations need not be followed by the user. The user may redefine the functions and designations of port signals to his device.

4) It is a NANODATA convention that the names of outbound signals, those going "from" the CPU, are prefixed by an "F" and inbound signals, those "to" the CPU, are prefixed by a "T".

SIGNALS FROM THE CPU

SIGNAL NAME / # OF LINES	LOGIC LEVEL / SIGNAL TYPE	SOURCE / DESTINATION	NOTES
FDS0-FDS5 / 6	HIGH / LEVEL	CPU G-BUS / ALL PORTS	Six data lines from the CPU normally used as a device selection code to route a command to a device.
FFUNC0-FFUNC5 / 6	HIGH / LEVEL	CPU PHANTOM BUS / ALL PORTS	Six data lines from the CPU normally used to specify a command.
FXIO / 1	LOW / LEVEL	XIO PRIMITIVE / KA SELECTED PORT	"Transmit I/O" is a programmable level Its length is determined by the number of consecutive T-periods in which the XIO primitive is active. It is normally used to select the channel to which data or command is sent.
FXIO STROBE / 1	LOW / PULSE	XIO PRIMITIVE / ALL PORTS	A strobe produced during every T-step in which the XIO primitive is specified. Normally used to strobe data/commands. Its timing is such that FDS, FFUNC, and FDATA are valid during XIO strobe.
FRIO / 1	LOW / PULSE	RIO PRIMITIVE / KA SELECTED PORT	"Read I/O" is used to gate the TDATA lines of the KA selected port into its port register and normally used to notify the channel/device that data has been read.
FIO CLK / 1	LOW / PULSE	CPU CLOCK / ALL PORTS	A synchronizing pulse produced once every T-step independent of any I/O controls.

FMC	LOW	CPU "MASTER CLEAR"	General master clear to all QM-1 hardware.
1	LEVEL	ALL PORTS	
FDATA0 - FDATA17	HIGH	EACH PORT REGISTER	The FDATA lines are the direct outputs of the port registers. There are eight sets of FDATA lines, one set for each separate port.
18	LEVEL	EACH PORT	

SIGNALS TO THE CPU

SIGNAL NAME / # OF LINES	LOGIC LEVEL / SIGNAL TYPE	DESTINATION	NOTES
TDATA0 - TDATA17 / 18	LOW / LEVEL	EACH PORT REGISTER	The TDATA lines connect to the preset inputs of a port register. There are eight sets of TDATA lines, one set for each individual port. The FRIO signal must be used by the port interface to gate data on these lines.
TID0-TID5 / 6	LOW / LEVEL	IO ID AUX	Six data lines which may be read by the CPU with an AUX -> F REG transfer. There are eight sets of TID lines, one set for each port. The low 3 bits of KA selects the actual port whose TID lines will be read. These lines are normally used as a device ID to identify the interrupting device.
TATTN DATA IN / 1	LOW / PULSE	ONE OF 30 INTERRUPT FLAGS	A pulse on this line latches the external interrupt level to which it is connected. Normally used to signal that input data is available. One line for each port.
TATTN DATA OUT / 1	LOW / PULSE	ONE OF 30 INTERRUPT FLAGS	Same as TATTN DATA IN. Normally used to request data to be output.
TATTN STATUS / 1	LOW / PULSE	ONE OF 30 INTERRUPT FLAGS	Same as TATTN DATA IN. One line for each port. Normally used to signal a change in status of an external

A-2.3 ELECTRICAL SPECIFICATIONS OF THE PORT SIGNALS

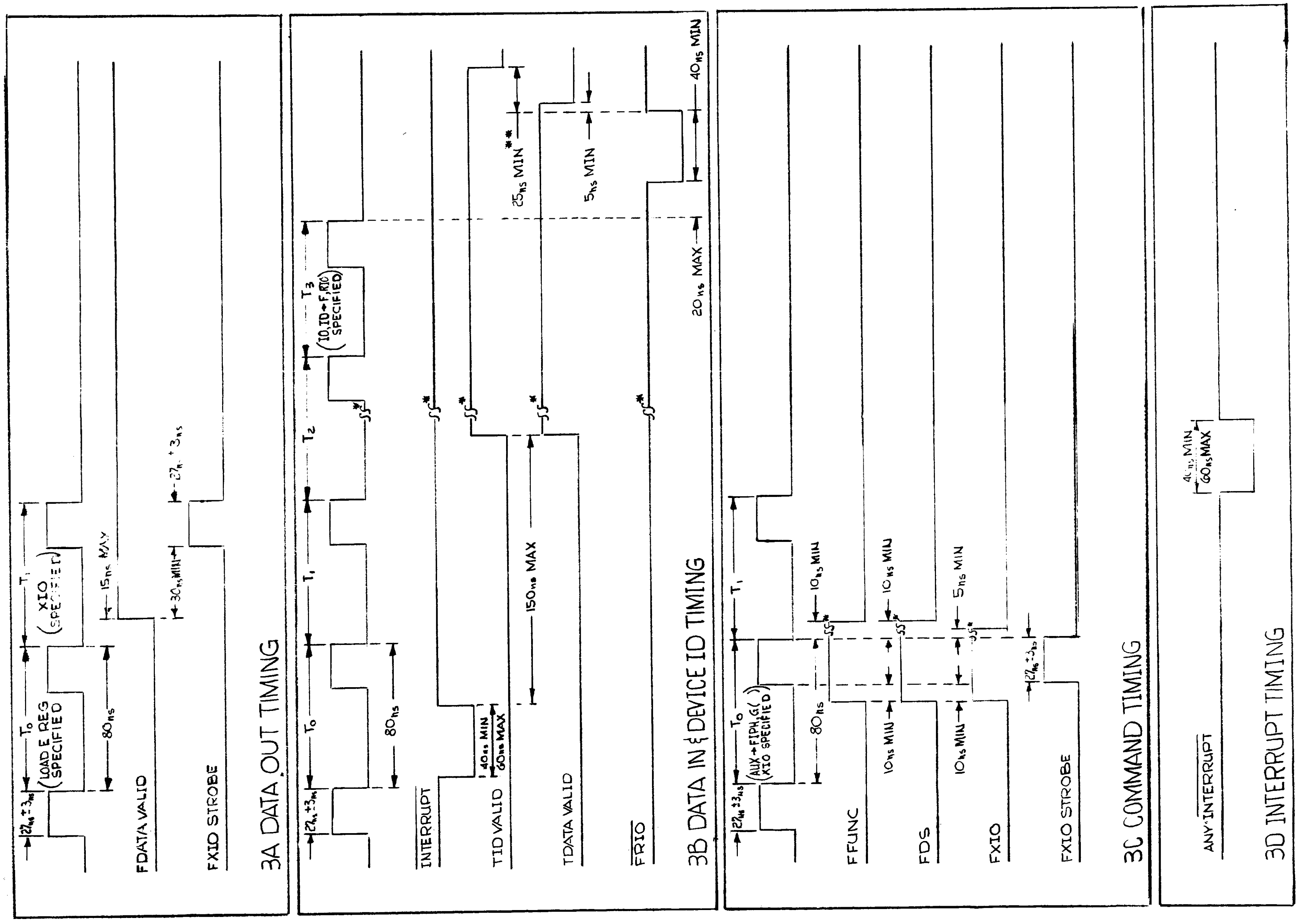
All drivers and receivers used by NANODATA are standard TTL, H or S series. Figure 2 describes the termination of signals at the QM-1 port. All signals must be buffered and terminated as shown by figure 1. Not more than one TTL (H series) load (2 ma) may be drawn from any port output. Electrical signals are defined as follows:

"HIGH" voltage	- Greater than 2.7 V, less than 5.5 V.
"LOW" voltage	- Less than 0.4 V, not less than -1.0 V.

These are values supplied by the port drivers, and interface drivers must conform to them. A space is provided for buffering logic on the interface card described in detail in section A-3.

Important note:

Resistor terminations shown in figure 2 with an asterisk must be installed on the port 0 interface card. They should not be used on any other port. If port 0 is not used, an interface card with terminating resistors must be installed in the port 0 connector.



NOTE:
∫∫* PROGRAM DEPENDENT
** TID HOLD TIME IS REFERENCED TO RIO SINCE NO POLL SIGNAL INDICATED IO,ID → F TRANSFER

FIG 3

A-2.4 TIMING OF THE PORT SIGNALS

Figure 3 describes the timing of the port signals related to the operations the signals are involved with.

NOTES:

Figure 3A: T1 is the earliest T-step where XIO can be specified if the data loaded into the port register is to be valid.

Figure 3B: "Interrupt" in figure 3B may be DATA-IN, DATA-OUT, or STATUS. Interrupt pulse timing is independent of CPU timing and may occur at any time. Data and device ID must be valid 150 nano-seconds from the trailing edge of the interrupt pulse, since that is the minimum time required for the CPU to respond to the inter-rupt. TDATA must remain valid for at least 5 nanoseconds after the trailing edge of RIO. Since there is no indication at the port that the ID is being read by th CPU, its hold time is referenced to RIO. Normally, RIO will release the data and ID.

Figure 3C: FXIO STROBE timing is such that all 3 elements of the command (function, device select, and XIO) can be specified in a single T-step.

Figure 3D: An interrupt of any type can occur at any time (with relation to the CPU timing). The pulse width must conform to the timing limits shown.

A-2.5 PIN ASSIGNMENTS OF THE PORT SIGNALS

TDATA(L)	0	- A3	I	FDATA	0	- A32	I	TID(L)	0	- C6
"	1	- C3	I	"	1	- C32	I	"	1	- A6
"	2	- A10	I	"	2	- A33	I	"	2	- C29
"	3	- C10	I	"	3	- C33	I	"	3	- A29
"	4	- A19	I	"	4	- A35	I	"	4	- C21
"	5	- C19	I	"	5	- C35	I	"	5	- A21
"	6	- A27	I	"	6	- A42	I	FDS	0	- C39
"	7	- C27	I	"	7	- C42	I	"	1	- A39
"	8	- A4	I	"	8	- A44	I	"	2	- C48
"	9	- C4	I	"	9	- C44	I	"	3	- A48
"	10	- A11	I	"	10	- A45	I	"	4	- C47
"	11	- C11	I	"	11	- C45	I	"	5	- A47
"	12	- A20	I	"	12	- A36	I	FFUNC	0	- C49
"	13	- C20	I	"	13	- C36	I	"	1	- A49
"	14	- A28	I	"	14	- A37	I	"	2	- C41
"	15	- C28	I	"	15	- C37	I	"	3	- A41
"	16	- A5	I	"	16	- A38	I	"	4	- C40
"	17	- C5	I	"	17	- C38	I	"	5	- A40

FRIO(L)	- A15	FXIO STR	- A31
ID SELECT(L)	- A16	FIO CLK	- A14
TATTN STATUS(L)	- A12	FMC(L)	- C14
TATTN DATA IN(L)	- C12	TATTN DATA OUT(L)	- C23

+ 5 V	- A9, A26, A43
- 5 V	- A7, C7
GND	- C1, A18, A34, C52

A-3 THE INTERFACE CARD

The QM-1 CPU has eight, 156 pin, female connectors each representing a port. The interface card is a NANODATA standard, UN series, wirewrap board with a 104 pin male connector designed to mate with a port connector in the CPU. This card simplifies the user's task of interfacing to the port by allowing him to put his drivers, receivers, and any logic directly in the Qm-1. The interface card is available in three different configurations:

UN 2	-	Has space for up to 30 14-pin or 16-pin DIPS and 8 24-pin DIPS.
UN 3	-	Has space for up to 12 14-pin or 16-pin DIPS and 16 24-pin DIPS.
UN 5	-	Has space for up to 48 14-pin or 16-pin DIPS.

Each configuration has a voltage plane, ground plane, 104 pin connector, and room for 3 44-pin cable connectors (Continental, MMM 44). +5 volt and -5 volt supplies are available at the port connector (see section A-2.5). Not more than 2 Amps may be drawn from the +5 volt supply and not more than 400 ma from the -5 volt supply.

If the user desires, NANODATA can supply interface cards completely assembled, wired, and tested to user specifications.

```
                 _             _             _
                / \           / \           / \
                ! !           ! !           ! !        /-----
                ! !           ! !           ! !       <<<<<     CABLES TO USER'S EQUIPMENT
                ! !           ! !           ! !        \-----
                ! !           ! !           ! !
              __! !__       __! !__       __! !__
         ____/_______\_____/_______\_____/_______\____
 I       I   I_______I     I_______I     I_______I   I
 N       I                                           I
 T       I   \_______________     _______________/   I
 E       I                   \ /                     I
 R       I                    V                      I
 F       I          44 PIN CABLE CONNECTORS          I
 A       I                                           I
 C       I    __________________________________     I
 E       I   I                                  I    I
         I   I   U S E R ' S   B U F F E R S    I    I
 C       I   I                                  I    I
 A       I   I             A N D                I    I
 R       I   I                                  I    I
 D       I   I           L O G I C              I    I
         I   I                                  I    I
 L       I   I                                  I    I
 A       I   I                                  I    I
 Y       I   I                                  I    I
 O       I   I__________________________________I    I
 U       I                                           I
 T       I         104 PIN PORT CONNECTOR            I
         I          "    "    "    "    "    "       I
          TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
```

PORT CONNECTOR POSITIONS ON QM-1 BACKPLANE

```
POSITION #    44   43   42   41   40   39   38   37

           ------------------------------------------------
           [ ____ ____ ____ ____ ____ ____ ____ ____     /
           [ I  I I  I I  I I  I I  I I  I I  I I  I     \
           [ I  I I  I I  I I  I I  I I  I I  I I  I     /
           [ I  I I  I I  I I  I I  I I  I I  I I  I     \
           [ I  I I  I I  I I  I I  I I  I I  I I  I     /
           [ I  I I  I I  I I  I I  I I  I I  I I  I     \
   ROW H   [ I  I I  I I  I I  I I  I I  I I  I I  I     /
           [ I  I I  I I  I I  I I  I I  I I  I I  I     \
           [ I  I I  I I  I I  I I  I I  I I  I I  I     /
           [ I  I I  I I  I I  I I  I I  I I  I I  I     \
           [ I  I I  I I  I I  I I  I I  I I  I I  I     /
           [ I  I I  I I  I I  I I  I I  I I  I I  I     \
           [ I  I I  I I  I I  I I  I I  I I  I I  I     /
           [ I  I I  I I  I I  I I  I I  I I  I I  I     \
           [ ---- ---- ---- ---- ---- ---- ---- ----     /
           ------------------------------------------------
    PORT #     0    1    2    3    4    5    6    7
```

APPENDIX B - QM-1 CPU OPTIONAL FEATURES

B-1 CONTROL STORE ADDRESS TRANSLATION AND ACCESS PROTECTION

B-1.1 GENERAL DESCRIPTION

The standard Control Store of the QM-I is a fully readable/writable 18-bit wide store, implemented in semiconductor memory. It is available in blocks of 1K words, up to a maximum of 16K words. Control Store is primarily used to hold microprograms and their associated tables and work areas. It is, however, a completely general-purpose memory and may be used in any way appropriate to a specific application. For example, Control Store is ideal for use as a scratch pad or cache memory.

In normal operation, Control Store is addressed from zero to the maximum installed memory address. Addresses beyond this range generate zeros for the "READ CS" command, and cause a null operation for the "WRITE CS" command. Any executing microprogram has access to the full range of installed addresses. All addresses must be absolute and no portion of control store may be excluded. For many applications, this mode of Control Store operation is sufficient. In particular, users executing a single microprogram stream will probably have no need for a more powerful mode of Control Store operation. For applications that do require a more powerful Control Store facility, the Control Store Address Translation and Access Protection Option is available.

With this option installed, microprograms have available a Virtual Address Space of 128K for Control Store. Translation hardware maps the address supplied by an executing microprogram into the actual address space of 16K (maximum) that corresponds to the Control Store physically installed on the machine. The actual address space is divided into 512 word pages; 32 such pages exist in the maximum configuration. The virtual address space is also divided into 512 word pages; 256 such pages may be referred to by a micro-program.

The translation between virtual page and actual page is accomplished by a small high-speed associative memory called the Associative Page Selector. This auxiliary memory is loaded by a control program when Control Store is initially written prior to releasing control to the currently executing microprogram. Now Control Store addresses are independent of the actual page address at which the page is loaded into Control Store. An executing microprogram can reference any of its currently loaded pages. No changes are necessary in the nano-primitives used by the microprogram.

Another auxiliary memory called the Page Access Control Memory is provided to permit control over which pages are accessable to the currently executing microprogram. This memory is also loaded by a control program prior to initiation of microprogram execution. The Page Access Control Memory contains a 2 bit status code for each physical page of Control Store loaded and each of 16 possible partition numbers. The code may be set to specify:

No Access Allowed
Read Access Allowed
Write Access Allowed
Full Access Allowed

In this way, Control Store may be "partitioned" as appropriate to the task now being executed. By a change in the 4 bit partition no. in FUSR, the executing environment may be completely changed.

Whenever Control Store access to the specified page is restricted for the current partition number, or whenever the Control Store Page referenced is not physically present, an addressing exception is generated. This permits an operating system to take the appropriate action.

The next two sections treat the detailed operation of the Control Store Address Translation and Access Protection Option.

B-1.2 DETAILED DESCRIPTION OF OPERATION

The basic nanoprimitives for accessing Control Store are unchanged. Whenever a "READ CS" or "WRITE CS" command is encountered, the 18 bit CS address which was generated by the CS Address Select Mechanism is actually loaded into the CS Address Buffer. The CS Address Translation Mechanism uses this buffer as the Virtual address and forms the physical address with which it accesses Control Store.

The 18-bit CS Address Buffer, along with a 4 bit "partition number", taken from the low order 4 bits of FUSR, provide the necessary inputs to the Control Store Address Translation and Access Protection hardware. Using these inputs, the hardware shown in Figure B-1.2a either permits access to the actual Control Store location desired or generates a Control Store addressing exception.

```
     FUSR                               18 Bit Control Store Address Buffer
 -------------                          -------------------------------------
 IX XI PART. I                          IXI      PAGE       I   DISPLACEMENT  I
 -------------                          -------------------------------------
  5 4  3 - 0           Bit               1 1       V      0 0        V       0
        bits         Numbers             7 6    8 Bits    9 8     9 bits     0
         V                                         V                 V
   -----------                             --------------            V
   I  PAGE   I                             I ASSOCIATIVE I           V
   I ACCESS  I                             I    PAGE     I           V
   I CONTROL I                             I  SELECTOR   I           V
   I MEMORY  I                             I             I           V
   I         I                             I    32 X 8   I           V
   I 16 x 64 I                             I             I           V
   -----------                             --------------            V
       V                                          V             -----------
    64 Bits                                    32 Bits          I CONTROL I
       V                                          V             I  STORE  I
       V                                   --------------       I         I
       V              READ CS ----->I      PAGE      I 32  I      32    I
       ----------------------------->I  VALIDATION   I>>>>>I    Pages   I
                      WRITE CS ---->I      LOGIC     I     I      of    I
                                           --------------       I   512   I
                                                  V             I  18-Bit I
                                               Address          I  Words  I
                                              Exception         -----------
```

Figure B-1.2A CONTROL STORE ADDRESS TRANSLATION AND ACCESS PROTECTION

The 4 bit partition number taken from the low order 4 bits of FUSR is input to a Page Access Control Memory (PACM) consisting of 16 by 64 bits. The 64 bits for each partition are actually 2, 32 bit registers, one READ Inhibit register and one Write Inhibit register, each bit with a single line to its respective physical Control Store page. These 2 bits allow or inhibit access to each page depending on their value as follows:

WRITE/READ	
1/1	- No access allowed
1/0	- Read access allowed
0/1	- Write access allowed
0/0	- Both Read and Write access allowed

At the same time, the 8 bits of the Control Store Address shown in Figure B-1.2A are input to the Associative Page Selector (APS). These 8 bits specify the Virtual Page number of one of the possible 256 virtual pages. The 32 by 8 bit associative memory provides a selection from one, or more, of the 32 possible physical pages of control store or indicates that the page is not available by selecting no page.

Finally, the "READ CS" or "WRITE CS" command, along with the 64 validation bits from the PACM and the 32-page-select bits from the APS are input to the Page Validation Logic Unit shown. If the validation bits show that the specified access is allowed, the page select is sent to Control Store and the low order 9 bits of the Control Store Address simultaneously select the word within that page.

If the access is inhibited or if no page select is available (indicating that the page is not present in Control Store), an addressing exception is generated.

Three generel AUX ACTION Commands are associated with the Control Store Address Translation Option. They are as follows:

63	SET ASSOCIATIVE MODE	- begins use of the associative translation and protection hardware.
64	SET DIRECT MODE	- turns off the use of the associative translation and protection hardware for the low 2K of physical Control Store.
65	LOAD CS ADDR BUFF	- loads the CS ADDRESS BUFFER with the 18 bit word pointed to by CS ADDR SELECT.

B-1.3 LOADING AUXILIARY MEMORIES

Both auxiliary memories involved in this option are loaded by means of an AUXILIARY ACTION command (see section 5.8.2). This permits an effective control over those microprograms that should not have the ability to modify the contents of the auxiliary memories.

The two main AUX ACTION commands are LOAD APS (60) and LOAD PAC(61). Both use the DIRECT CS ADDR (the 18 bit word pointed to by the CS ADDR. SELECT in the T-Vector) and LOAD APS(60) uses the contents of the CS ADDR. BUFFER. The interpretation of each memory load command is shown below.

ASSOCIATIVE PAGE SELECT MEMORY

```
     Specified by CS Addr Select         Loaded by LOAD CS ADDR BUFF(AUX ACT 65)
         DIRECT  CS ADDRESS                          CS ADDRESS  BUFFER
----------------------------------------             ----------------------
I HI 16/ LO 16/ BINARY - CELL SelectionI             IX/  DATA  /XXXXXXXXXI
----------------------------------------             ----------------------
    1      1            16                   Bits     1     8        9
```

One or more cells of the 32 available cells are loaded with the 8 DATA bits which are currently resident in the CS Address Buffer (placed there by the LOAD CS ADDR BUFF - AUX ACTION 65, or the last READ or WRITE CS command). Which cells are loaded is decided by the 18 bits of data called the DIRECT CS ADDRESS (currently pointed at by the CS ADDR SELECT). Bit 17 on (1) choses the HIGH 16 cells, and bit 16 on (1) choses the LOW 16 cells; bits 15 thru 0 are a binary selection of cells 15 - 0 (mod 16).
For Example: 400002 selects cell 17 ; 200003 selects cells 0 and 1.
THAT IS: AUX ACTION 60 causes 8 bits of data from the CS ADDR BUFFER to be written to n cells (1 < n < 32) of the Assoc. Page Select Memory, selected from the DIRECT CS ADDR pointed to by the CS ADDR SELECT.

AUX ACTION 62 - READ APS uses the DIRECT CS ADDR in the same way.

PAGE ACCESS CONTROL MEMORY

```
        DIRECT CS ADDRESS (pointed to by CS ADDR SELECT)
   ----------------------------------------------------------
   I WRITE-READ / HIGH-LOW/ 16 bit INHIBIT(1) or  ALLOW(0) I
   I half pair  / half reg/          for 16 pages          I
   ----------------------------------------------------------
 bits    17            16        15                         0
```

The LOAD PAC Command (AUX ACTION 61) is used to load Page Access Control flags. LOAD PAC loads sixteen flags at a time, changing only one user's READ or WRITE access authorization as identified by FUSR. The low 16 bits of the DIRECT CS ADDRESS (pointed to by CS ADDR SELECT) are the flags to be loaded. Bit 17 specifies whether the access flags are for READ or WRITE authorization, while bit 16 specifies the upper or lower half of 32 pages being given authorization:

00	=	WRITE HIGH HALF
01	=	WRITE LOW HALF
10	=	READ HIGH HALF
11	=	READ LOW HALF

THAT IS: AUX ACTION 61 LOADS PAC (one half of one register) belonging to user (FUSR) from DIRECT CS ADDRESS inhibiting(or allowing) access to 16 pages at once.

B-1.4 APPLICATIONS

The control Store Address Translation and Access Protection Option is designed for those sophisticated applications that require a more powerful memory capability at the microprogramming level. The option:

1. Permits sharing of re-entrant microcoding.

2. Allows all concurrent micro-processes to have independent address spaces.

3. Provides basic Control Store Program Protection.

4. Provides Control Store Partitioning between 15 independent tasks.

5. Facilitates efficient, high speed, switching between resident microprocesses.

For those applications that either require, or can make effective use of, one or more of these capabilities, the option should be installed.

Comments regarding errors, deficiencies, or omissions in this document will be appreciated. Comments should be sent in writing to the Technical Services Manager, NANODATA CORPORATION, 2457 Wehrle Drive, Williamsville, New York 14221.