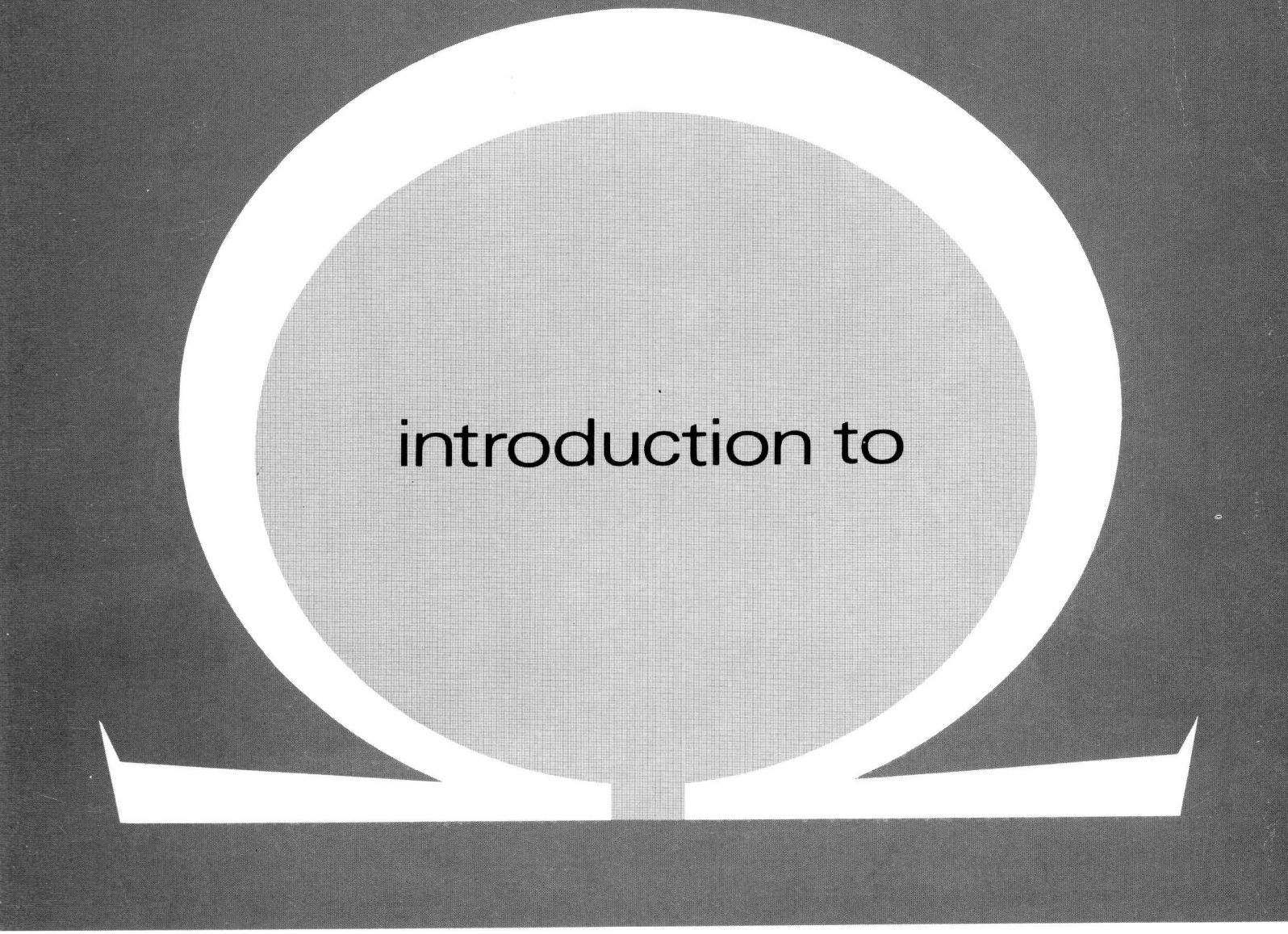
№ 028

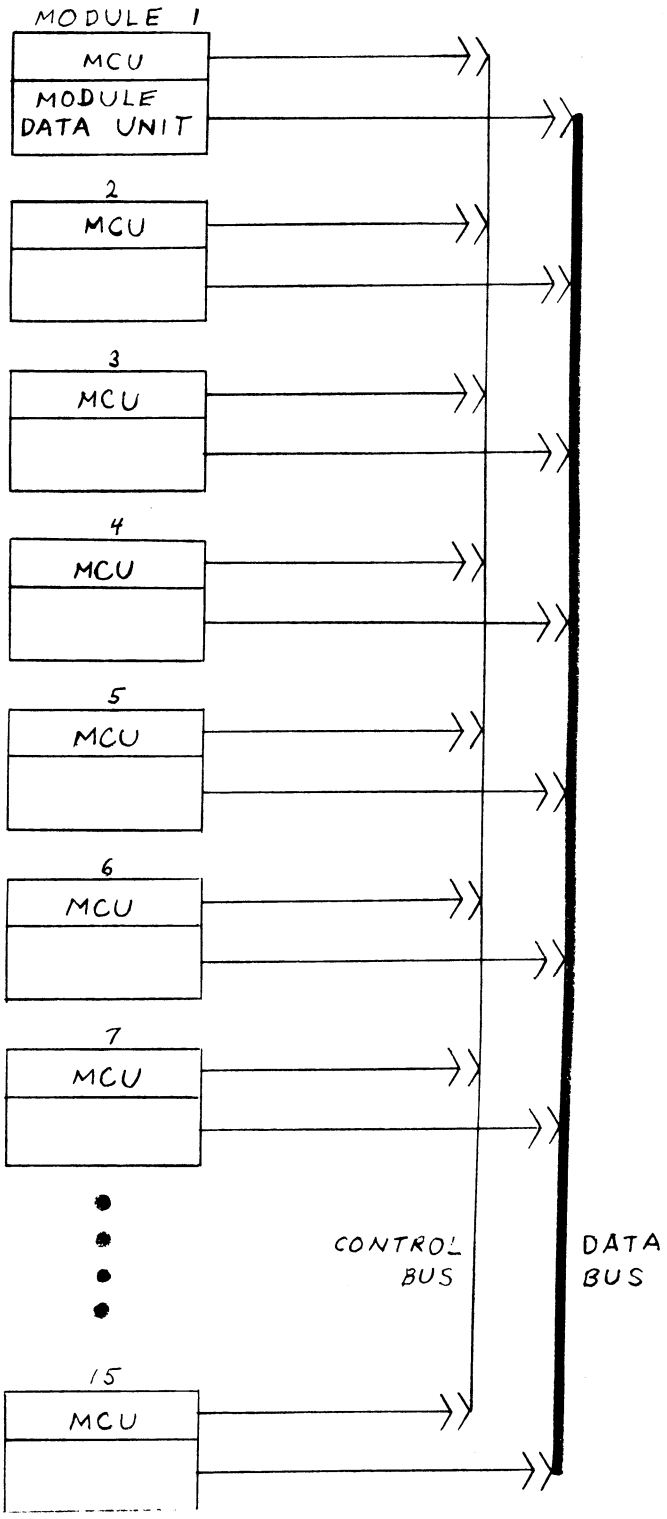
HEWLETT hp PACKARD



OMEGA

company private

Omega Hardware
TABLE OF CONTENTS


MODULE 1
MCU
MODULE DATA UNIT
2
MCU
3
MCU
4
MCU
5
MCU
6
MCU
7
MCU
15
MCU
CONTROL BUS
DATA BUS

HP OMEGA

"HP Omega" specifies a new line of Hewlett-Packard computers. This introductory manual defines a specific model of this new line: Omega-32.

Omega-32 is a 32-bit system, designed to cover a large range of computing power. It is not to be construed specifically as a large system or a small system.

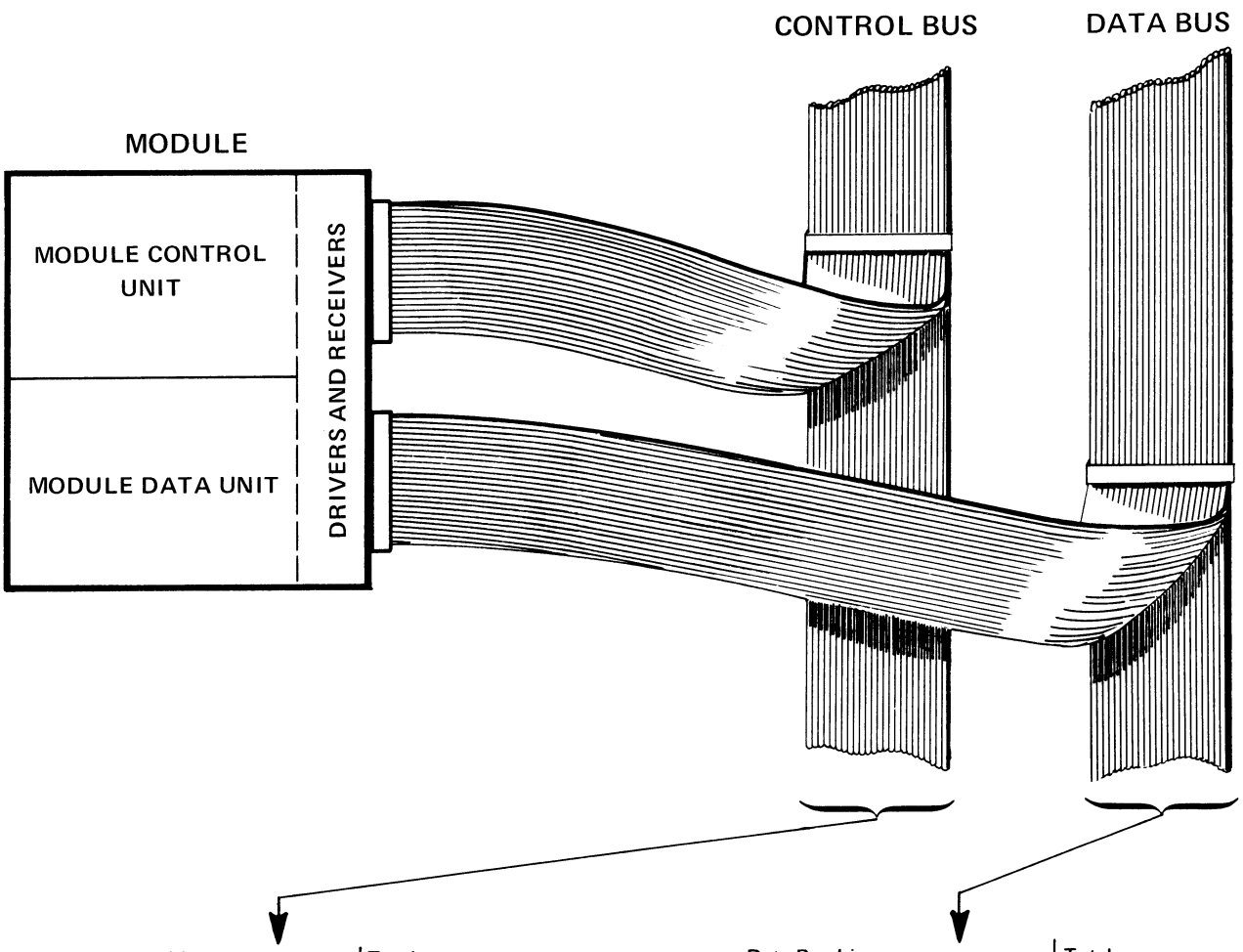
To accomplish this large range of power, Omega utilizes a fully modular approach. This means simply that we have independent, asynchronous plug-in modules that communicate with each other by way of a common Data Bus. Intermodule commands and responses are transmitted through a Control Bus. Operations within the modules may be proceeding in parallel, at their own rates.

A module is any unit that may require data communication with another unit. This includes arithmetic processors, memory units, I/O processors, or any future special-purpose processors. The module mix determines the size, cost, and performance of the system. Up to four Central Processor Units can operate simultaneously in the Omega system.

The system can accommodate a maximum of 15 modules. Modules are numbered 1 through 15. These numbers are assignable by the user, and easily changed by moving plug-in components within each module. This number becomes the module's "Module Address".

Each module includes a Module Control Unit (MCU) consisting of six PC cards. Data is gated in or out of the modules under control of the MCU.

The "Module Data Unit" is a general term used in this text to denote the primary functional part of a module excluding the MCU. This may be memory control logic, central processor logic, etc.



Control Bus Lines	Total
LOINH	14
HIINH	15
BUSY	15
CPINH	4
Clock	1
	49

Data Bus Lines	Total
Data Bits (0-31)	32
Parity Bits (data)	4
FROM	4
TO	4
To-From Parity Bit	1
Power On	1
Power Fail	1
System Parity Error	1
	48

MNEMONICS

Above:	LOINH	Low Inhibit
	HIINH	High Inhibit
	CPINH	Central Processor Inhibit
Others:	DES	Destination
	MADD	Module Address
	SL	Select
	SLEN	Select Enable
	LORQ	Low Request
	HIRQ	High Request
	MRQ	Multiple Request
	RY	Ready to Receive

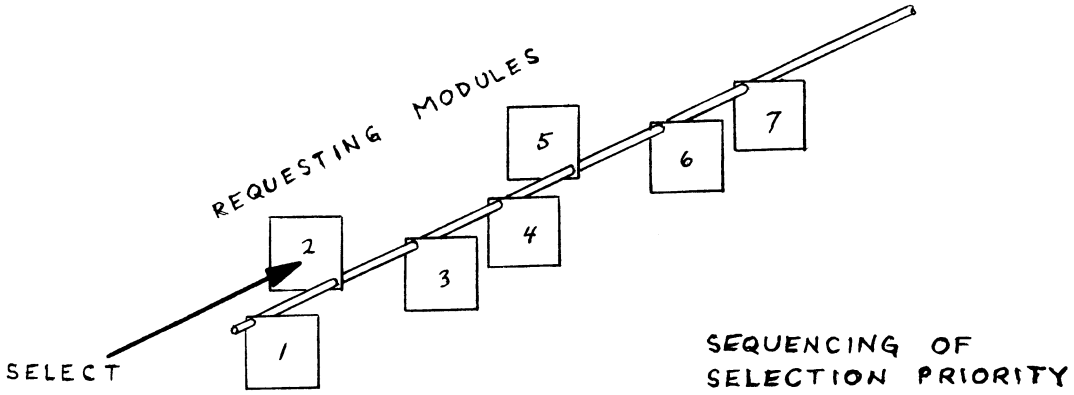
THE BUS SYSTEM

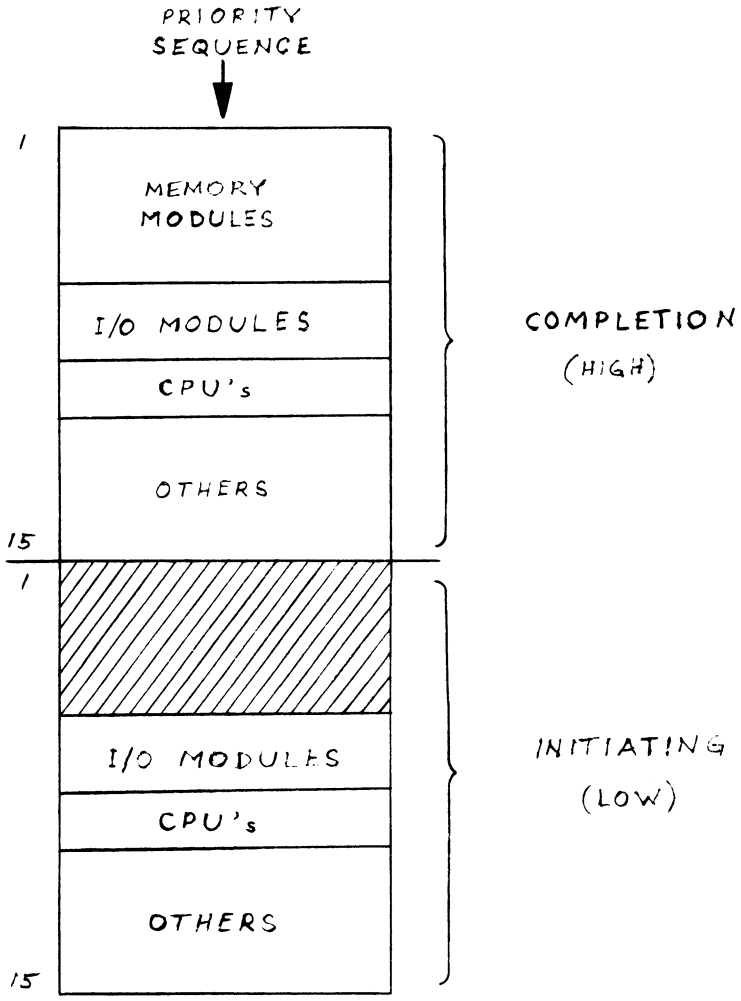
As indicated in the previous diagram, the Data Bus connects to the Data Unit of each module, and the Control Bus connects to the Control Unit in each module. The connection, however, is not direct. Each incoming and outgoing line goes through a driver or receiver in the module. Since there are a total of 97 lines, 97 drivers and 97 receivers are required. These occupy the front portions of the two MCU cards. The rear portions of the MCU cards contain the bus control logic; this arrangement reduces the need for back-plane signal routing.

As shown at left, short cable stubs from the module tap directly onto the buses, which run through the center of the system. (The physical routing is illustrated later.) In addition to the 97 active lines, each alternate line is a ground line to minimize interference, so there are actually 194 conductors in the buses and mating stubs.

The system Clock is generated in one of the system's MCU's and is distributed on the Bus to the other MCU's. Since the physical bus distance from the Clock generator to each module is not uniform, the Clock is variably delayed within each module, so that the modules will operate on the same reference times. The Clock cycle time is 100 nanoseconds. It is a design rule that all control and storage elements for the Bus may change state only on the leading edge of the system Clock.

The bus lines listed on the facing page will be explained in subsequent descriptions. With reference to the mnemonics, a parenthetical number will frequently be used to indicate one specific line of a group. Thus, BUSY(6) means the Busy line from Module 6.


REQUESTING MODULES
1
2
3
4
5
6
7
SELECT
SEQUENCING OF
SELECTION PRIORITY


PRIORITY
SEQUENCE
1
MEMORY
MODULES
I/O MODULES
CPU's
OTHERS
15
1
I/O MODULES
CPU's
OTHERS
15
COMPLETION
(HIGH)
INITIATING
(LOW)

MCU FUNCTIONS

The fact of having a common Data Bus necessitates a means of deciding which module gets the Bus when there are several simultaneous requests to transmit.

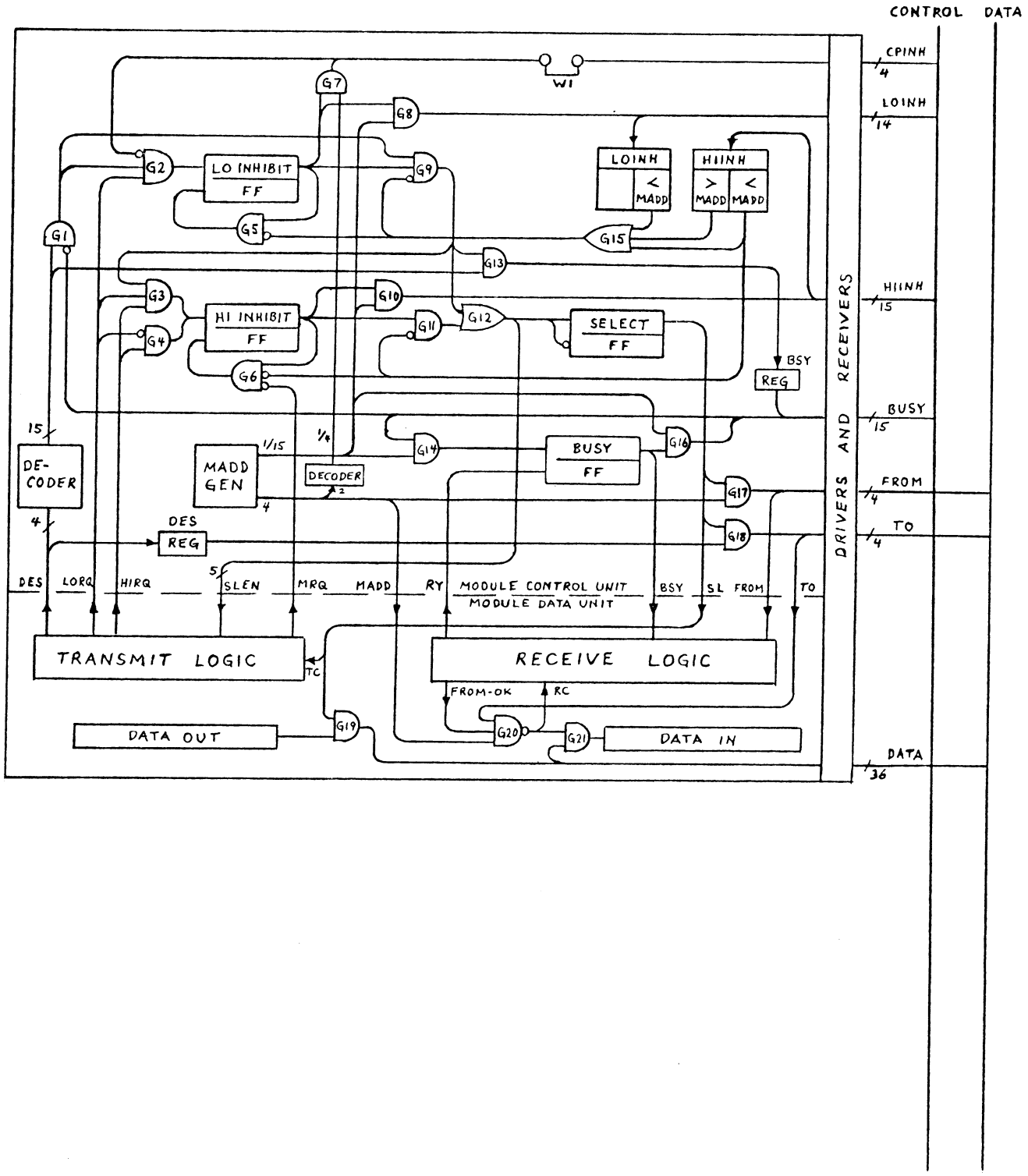
It is the function of the Module Control Unit (MCU) to establish the priority of transmission.

Priority might be visualized as a series of flags that can be raised to a Request position, as shown in the upper diagram at left. The raised Request with the lowest module number is the first in view, blocking or "inhibiting" those behind it. Module 2 in the diagram will be the first selected, then (unless 1, 3, or 4 comes up at this time) Module 5 will be next. Requests can be raised on any Clock cycle, so priority checking is continuous in the MCU's.

There are two levels of priority: high and low. See lower illustration. High priority is categorized as a "completion" transmission, and Low priority is categorized as an "initiating" transmission. This means that a transmission that completes some earlier action (for example, returning data from memory that was requested earlier) always has priority over starting a new operation. Since memory can never initiate an operation by itself, its transmission requests are always "completions", i.e., high priority. CPU's and I/O Processors can both initiate and complete.

For reasons of priority, Module Addresses are ordinarily assigned in the following order: memory modules, then I/O Processor modules, then Central Processor modules. The reasoning is that memory transfers are most important because some other module will be waiting for its data. I/O Processors are next most important, since they operate with real-time devices, which may have very fast synchronous data transfer rates. CPU's, on the other hand, operate in machine time, and execution delays of a few cycles — or many — are usually of little consequence.

Although priority is primarily dependent on the Module Address, there is one exception. In systems with more than one Central Processor, none has fixed priority over the other; the first one to request will be the first one selected. Apart from this exception, priority descends from Module 1 Completion (highest) to Module 15 Initiating (lowest).


CONTROL
DATA
W1
G1
G2
G3
G4
G5
G6
G7
G8
G9
G10
G11
G12
G13
G14
G15
G16
G17
G18
G19
G20
G21
LO INHIBIT FF
HI INHIBIT FF
LOINH
< MADD
HIINH
> MADD
< MADD
SELECT FF
BSY REG
BUSY FF
MADD GEN
1/15
DECODER
1/4
2
DE-CODER
15
4
DES REG
4
5
DES
LORQ
HIRQ
SLEN
MRQ
MADD
RY
MODULE CONTROL UNIT
MODULE DATA UNIT
BSY
SL
FROM
TO
TRANSMIT LOGIC
TC
RECEIVE LOGIC
FROM-OK
RC
DATA OUT
DATA IN
DRIVERS AND RECEIVERS
CPINH
4
LOINH
14
HIINH
15
BUSY
15
FROM
4
TO
4
DATA
36

MCU LOGIC

The following pages will describe the MCU operation. First, however, note the major logic elements of the MCU on the facing page.

The Module Control Unit itself consists of the elements above the dashed line, plus the Drivers and Recievers for the Bus lines along the right edge of the diagram. Signal lines going to and from the Module Data Unit are unidirectional, marked with arrowheads where they cross the dashed line. Signal lines going to and from the Buses are bidirectional, coming in through a Receiver circuit and out through a Driver.

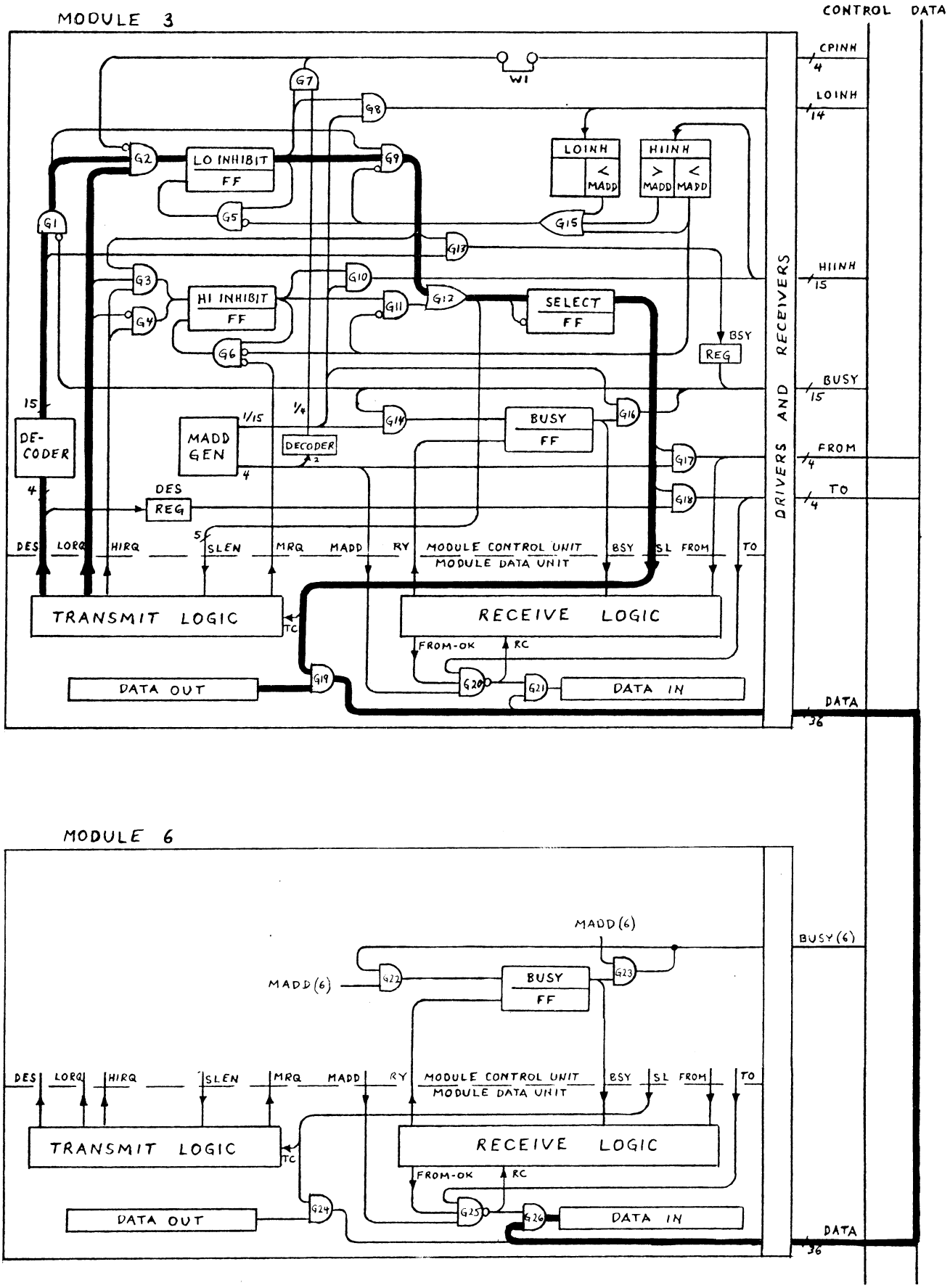
Transmit and Receive Logic in the Data Unit are shown as generalized blocks here. Actual logic will vary from module to module. There may be, for example, more than one DATA IN or DATA OUT registers.

The Module Address Generator (MADD GEN) continuously supplies two forms of the Module Address: a 4-line binary output, and a selected single-line output (1 of 15). The address is generated by a hardwired adapter board, unique for each module, into which the MCU control cards are seated. The 4-line output identifies the transmitting module (FROM code, to the Bus via G17), and controls inputs at G20 so only those transmissions intended for "this module" will be accepted. The single-line output selects one of a series of 15 gates, at four separate points (represented as 4 single gates: G8, G10, G14, G16). If, for example, "this module" is Module 6, G8 represents the 6th of 15 gates, enabling the Lo Inhibit flip-flop to control the LOINH(6) line on the Data Bus.

The four flip-flops indicate the state of the module:

LO INHIBIT: The module is making a low priority request to use the Bus.
HI INHIBIT: The module is making a high priority request to use the Bus.
SELECT: The module is selected to use the Bus.
BUSY: The module is not ready to receive data.

The sequence of events on the following pages illustrates a typical case: Module 3 initiates a transmission to Module 6, then Module 6 returns data to Module 3 with a completion transmission.


MODULE 3
CONTROL
DATA
W1
G7
G8
G2
LO INHIBIT
FF
G9
G5
G1
LOINH
< MADD
HIINH
> MADD
< MADD
G15
G13
G3
HI INHIBIT
FF
G10
G11
G12
SELECT
FF
G4
G6
BSY
REG
15
DE-
CODER
1/15
MADD
GEN
4
1/4
DECODER
2
G14
BUSY
FF
G16
G17
G18
4
DES
REG
5
DES
LORQ
HIRQ
SLEN
MRQ
MADD
RY
MODULE CONTROL UNIT
MODULE DATA UNIT
BSY
SL
FROM
TO
TRANSMIT LOGIC
TC
RECEIVE LOGIC
FROM-OK
RC
DATA OUT
G19
G20
G21
DATA IN
DRIVERS AND RECEIVERS
CPINH
4
LOINH
14
HIINH
15
BUSY
15
FROM
4
TO
4
DATA
36
MODULE 6
MADD(6)
BUSY(6)
MADD(6)
G22
BUSY
FF
G23
DES
LORQ
HIRQ
SLEN
MRQ
MADD
RY
MODULE CONTROL UNIT
MODULE DATA UNIT
BSY
SL
FROM
TO
TRANSMIT LOGIC
TC
RECEIVE LOGIC
FROM-OK
RC
DATA OUT
G24
G25
G26
DATA IN
DATA
36

INITIATING TRANSMISSION

1. Module 3 initiates a Low Request (LORQ) together with a 4-bit Destination address. This address (6, in this case) is stored in the DES register, and is decoded and compared with BUSY(6) at gate G1.

2. If G1 indicates Module 6 is not Busy, LORQ sets the Lo Inhibit flip-flop on the next Clock transition, via G2. (The CPINH input is a special case, discussed later; assume it is low.)

3. The Lo Inhibit output is gated with MADD(3) at G8, and is sent out on the Control Bus as LOINH(3).

4. If G15 detects no other LOINH with a module number less than 3, or no HIINH (Hi Inhibit) of any number, and if Module 6 is still not Busy, G9 enables the next Clock to set the Select flip-flop.

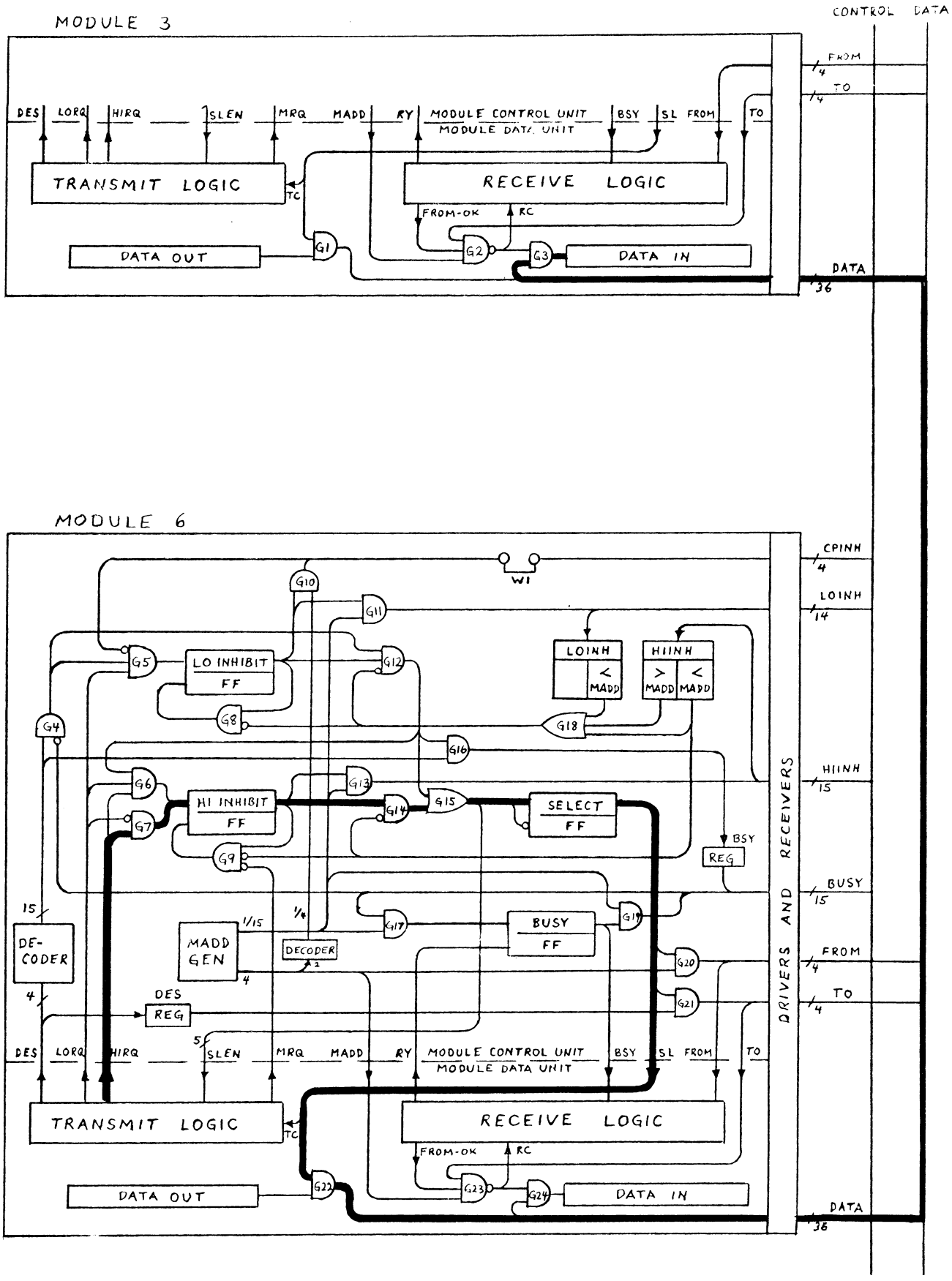
5. Since the transmission to Module 6 is now assured to occur on the next cycle, Module 3 immediately raises Module 6's BUSY line so that it will appear busy to all other modules. Module 3 does this via G13 and a BSY register (which holds the signal for one cycle, since DES may be removed as soon as selection occurs). BUSY(6) sets Module 6's Busy flip-flop via G22. (The Busy flip-flop will be cleared after Module 6 has received the transmission and issues Ready signal RY.)

6. Simultaneously (same Clock that sets Select), the set output of the Lo Inhibit flip-flop, fed back via G5, clears that flip-flop. However, if G15 had inhibited the setting of Select (at G9, in step 4), G15 would also inhibit G5 from clearing Lo Inhibit. This, in effect, would keep Module 3's request for priority (LOINH) active.

7. Select strobes out data, the FROM code, and the TO code at G17, G18, and G19. Select also signifies Tranmission Complete (TC) to the Transmit Logic.

8. All free modules clock the Bus data into their DATA IN register, since G25 (in all modules) normally supplies a high to G26. However, only in Module 6 does the TO code compare with MADD (at G25). So, providing there is no restriction on the FROM code (discussed further on the next page), G25 inhibits G26, thus locking in the data.

9. The low G25 output goes back to the Receive Logic, signifying Receive-Complete (RC). Module 6 now cannot be transmitted to until it has used the information in the DATA IN register, and clears the Busy flip-flop again with an RY signal.


MODULE 3
CONTROL
DATA
FROM
TO
DES
LORQ
HIRQ
SLEN
MRQ
MADD
RY
MODULE CONTROL UNIT
MODULE DATA UNIT
BSY
SL
FROM
TO
TRANSMIT LOGIC
TC
RECEIVE LOGIC
FROM-OK
RC
DATA OUT
G1
G2
G3
DATA IN
DATA
36
MODULE 6
CPINH
LOINH
HIINH
BUSY
FROM
TO
DRIVERS AND RECEIVERS
W1
G10
G11
G5
LO INHIBIT
FF
G12
G8
LOINH
< MADD
HIINH
> MADD
< MADD
G18
G4
G16
G6
G13
HI INHIBIT
FF
G14
G15
SELECT
FF
G7
G9
BSY
REG
15
DE-
CODER
MADD
GEN
1/15
1/4
DECODER
2
G17
BUSY
FF
G19
G20
G21
4
DES
REG
5
DES
LORQ
HIRQ
SLEN
MRQ
MADD
RY
MODULE CONTROL UNIT
MODULE DATA UNIT
BSY
SL
FROM
TO
TRANSMIT LOGIC
TC
RECEIVE LOGIC
FROM-OK
RC
DATA OUT
G22
G23
G24
DATA IN
DATA
36

COMPLETION TRANSMISSION

1. The high-priority Completion transmission to Module 3 implies that Module 3 is expecting the data, and so cannot accept transmissions from any module except Module 6. This is accomplished by having Module 3 keep its Busy flip-flop set, while applying a low FROM-OK to G2. The Receive Logic stores the number "6" in a register, and will not raise the FROM-OK line until the FROM bits on the Bus compare exactly with the contents of this register.

2. Module 6 issues a High Request (HIRQ) via G7 to set the Hi Inhibit flip-flop on the next Clock transition. Note that no check for receiver-busy is made. (The G6 input to the Hi Inhibit flip-flop is used for multiple transmissions, discussed on the following pages.) The Destination address is simultaneously stored in the Destination register.

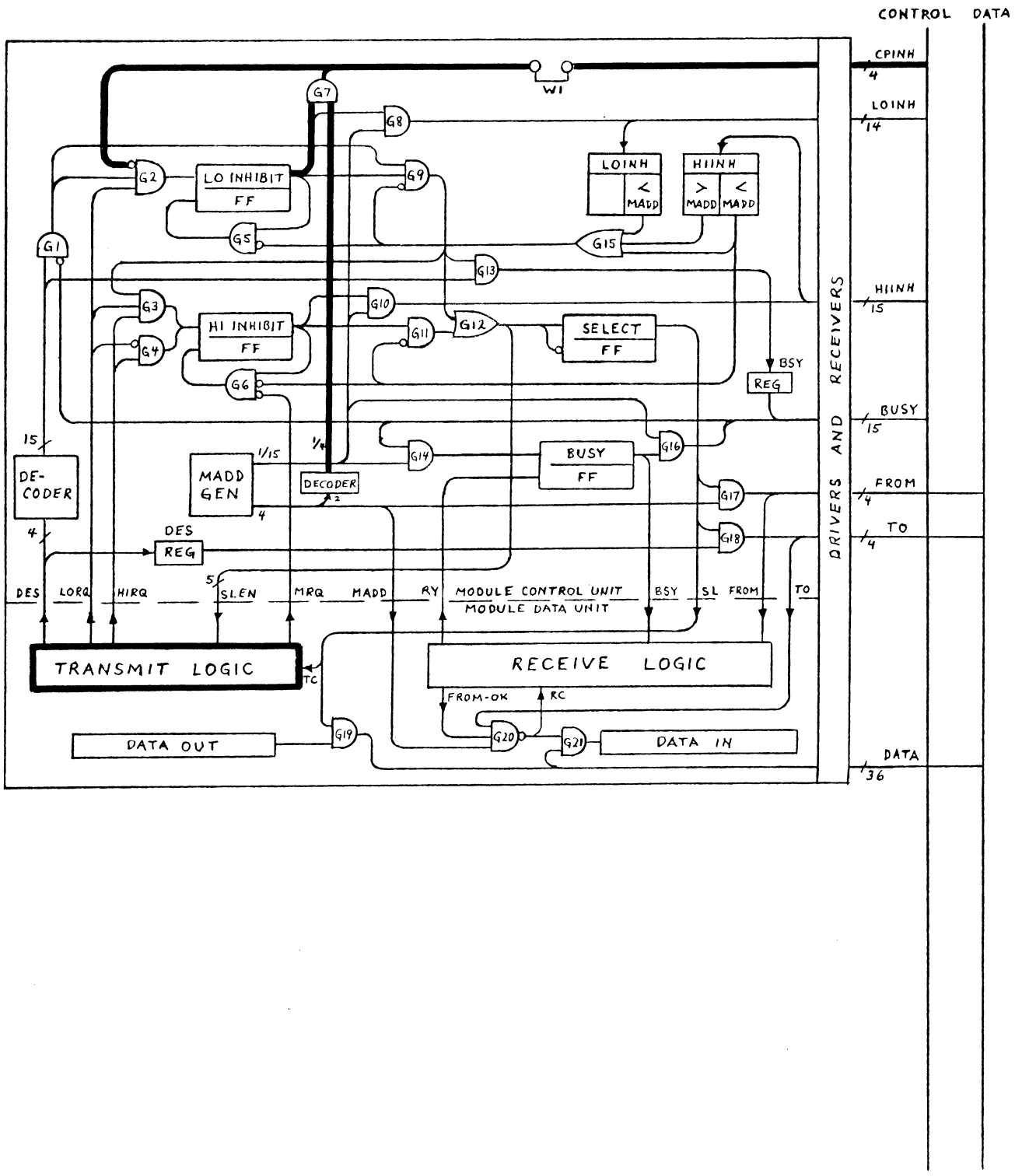
3. HIINH(6) is raised on the Bus via G13. If G14 detects no higher priority Inhibit (only HIINH with address less than 6), Select is set on the next Clock.

4. Simultaneously, the set output of the Hi Inhibit flip-flop, fed back via G9, clears that flip-flop. However, if a higher priority Inhibit had prevented setting Select (preceding step), or if this is to be a Multiple Request (MRQ), G9 would be prevented from clearing Hi Inhibit. This would keep Module 6's request for priority (HIINH) active.

5. Select strobes out data, the FROM code, and the TO code at G20, G21, and G22. The data is clocked into all free DATA IN registers, including that of Module 3.

6. In Module 3, the Receive Logic compares the FROM code on the Bus with the stored code (step 1), and raises FROM-OK. G2 also compares TO with MADD, and having all three inputs high, now inhibits G3, locking the data in the DATA IN register.

7. The low G2 output goes back to the Receive Logic, signifying Receive-Complete (RC). After Module 3 has used the information in its DATA IN register, it clears the Busy flip-flop with an RY signal.


CONTROL
DATA
W1
G1
G2
G3
G4
G5
G6
G7
G8
G9
G10
G11
G12
G13
G14
G15
G16
G17
G18
G19
G20
G21
LO INHIBIT FF
HI INHIBIT FF
SELECT FF
BUSY FF
LOINH
HIINH
MADD
REG
BSY
DE-CODER
MADD GEN
DECODER
DES REG
DES
LORQ
HIRQ
SLEN
MRQ
MADD
RY
MODULE CONTROL UNIT
MODULE DATA UNIT
BSY
SL
FROM
TO
TRANSMIT LOGIC
RECEIVE LOGIC
TC
FROM-OK
RC
DATA OUT
DATA IN
DRIVERS AND RECEIVERS
CPINH
LOINH
HIINH
BUSY
FROM
TO
DATA

SPECIAL MCU OPERATIONS

CPU REQUESTS

As mentioned before, the system can have up to four Central Processor Units, and none has fixed priority over the other. Instead, the following rules apply:

1. The first CPU to make a request for priority (Lo Inhibit set) will prevent all other CPU's from requesting priority until the first CPU is selected.

2. If more than one CPU make simultaneous requests for priority, they will be selected in order of priority. The other CPU's will be inhibited until the original group has been selected. (Lo Inhibit is stored until the module is selected.)

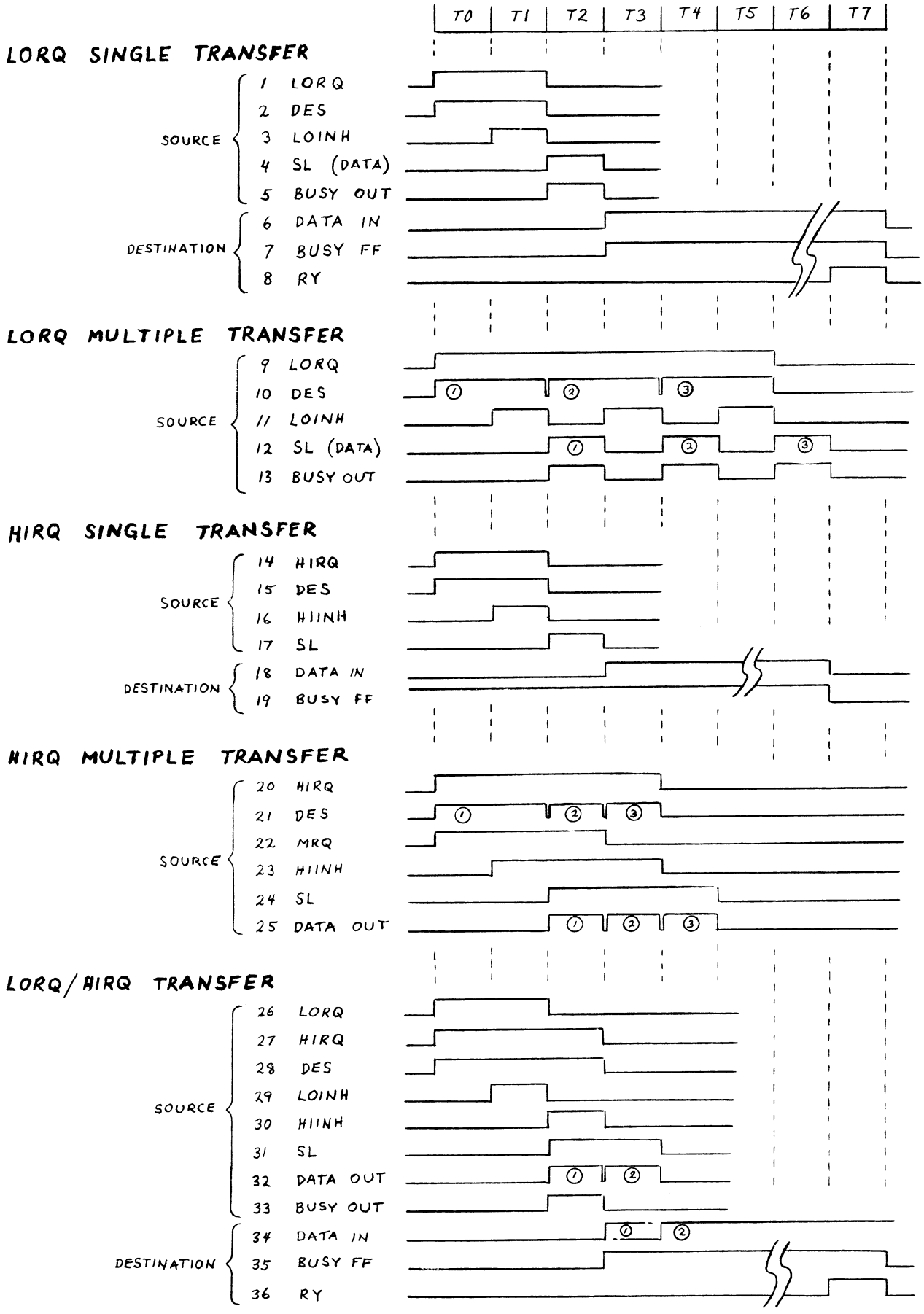
The logic is implemented by assigning each CPU to one of four CPINH lines, so that G7 in one CPU can inhibit G2 in the others. The assignment of lines to CPU's is made by decoding the two least significant bits of MADD, which necessarily means that the CPU's must have consecutive addresses (or 4 different combinations of the two least significant bits). Jumper W1 is connected only in CPU modules.

MULTIPLE TRANSFERS

The Module Data Unit (MDU) has 3 lines on which it can pull to provide for different types of transfers: either low or high priority, and single or multiple request — or some combination of these. The 3 lines are: LORQ (Low Request), HIRQ (High Request), and MRQ (Multiple Request). In addition, the MCU has 4 Destination lines, to determine where each transmission is to go.

Multiple transfers are accomplished by raising the desired combination of lines, and controlling the duration of each signal. The MDU has two inputs from the MCU that tell the MDU when it may change the control lines. These inputs are SLEN (Select Enable) and SL (Select). SLEN, actually consiting of the five lines that set Select (the inputs of G9 and G11), tells the MDU that its data will be strobed out on the next Clock. It may therefore proceed, for example, to enable another DATA OUT register in preparation for Multiple Transfer. SL tells the MDU that transmission is now occuring, and is complete at the end of that cycle.

There are 5 basic types of transfers, illustrated by timing diagrams on the following page. The time periods, T0 through T7, are for reference only, and are 100 nanoseconds each.


T0
T1
T2
T3
T4
T5
T6
T7
LORQ SINGLE TRANSFER
SOURCE
1 LORQ
2 DES
3 LOINH
4 SL (DATA)
5 BUSY OUT
DESTINATION
6 DATA IN
7 BUSY FF
8 RY
LORQ MULTIPLE TRANSFER
SOURCE
9 LORQ
10 DES
11 LOINH
12 SL (DATA)
13 BUSY OUT
HIRQ SINGLE TRANSFER
SOURCE
14 HIRQ
15 DES
16 HIINH
17 SL
DESTINATION
18 DATA IN
19 BUSY FF
HIRQ MULTIPLE TRANSFER
SOURCE
20 HIRQ
21 DES
22 MRQ
23 HIINH
24 SL
25 DATA OUT
LORQ/HIRQ TRANSFER
SOURCE
26 LORQ
27 HIRQ
28 DES
29 LOINH
30 HIINH
31 SL
32 DATA OUT
33 BUSY OUT
DESTINATION
34 DATA IN
35 BUSY FF
36 RY

TYPES OF TRANSFERS

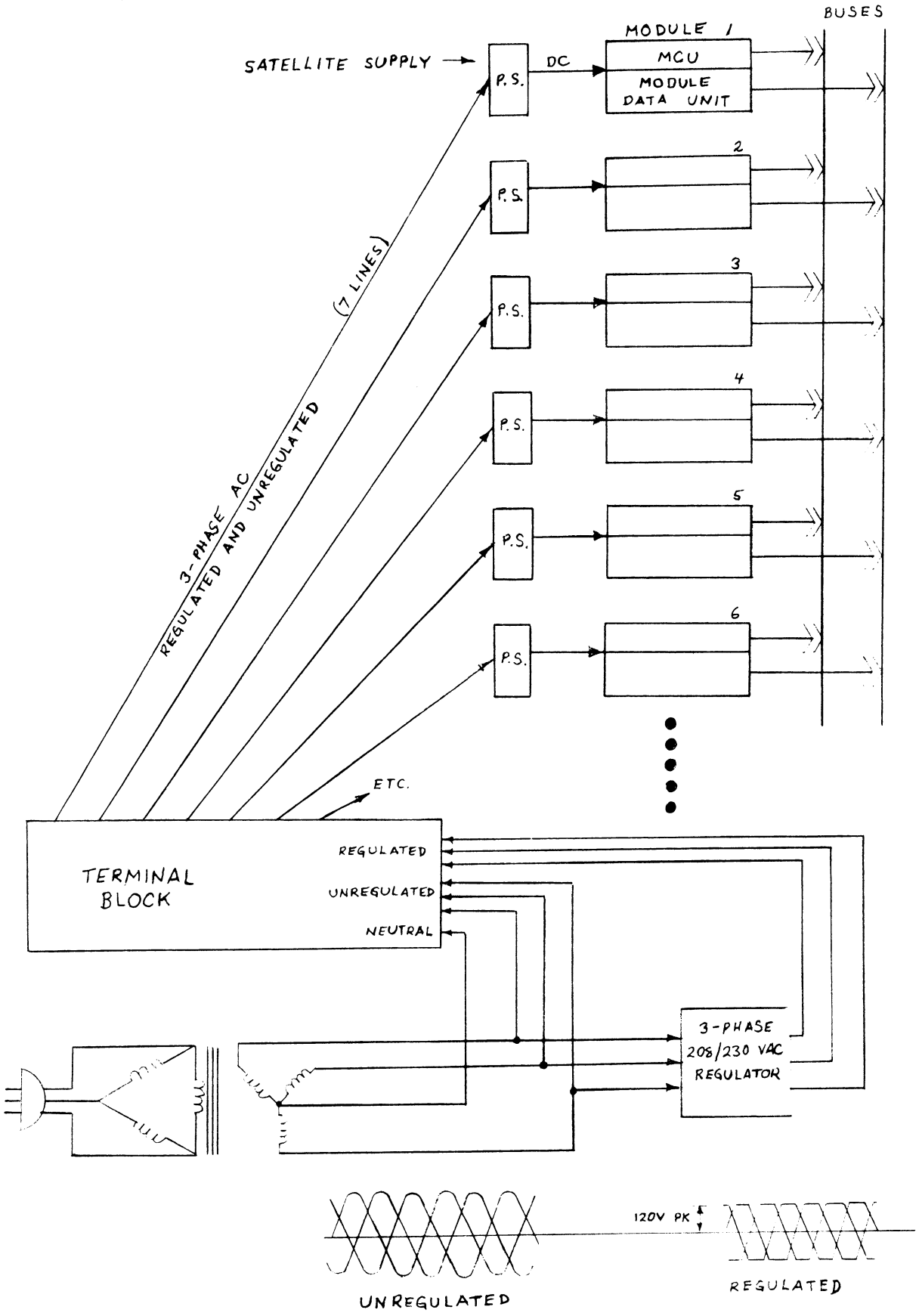
LORQ SINGLE TRANSFER. A Lo Request and Destination code are issued to MCU at T0. The Lo Inhibit sets at T1, during which time LOINH propagates through the system, and other Inhibits could arrive to inhibit Select. Assuming there is no higher-priority Inhibit, Select sets and strobes out the Data at T2, while SLEN turns off LORQ and DES. The Source also raises the BUSY signal of the Destination at T2, causing its Busy flip-flop to set at T3. Data propagates to the Destination during T2, and is clocked in at T3.

LORQ MULTIPLE TRANSFER. Instead of permitting SLEN to clear LORQ, as above, LORQ is maintained high for as many data Selects as desired. Note that SL occurs on alternate cycles. Each transfer is to a different Destination. A new DES address must therefore be supplied by the MDU on receipt of each SLEN. On the last SLEN, the MDU must turn off the LORQ. Concurrent with SL, the Source raises the BUSY line of each addressed destination.

HIRQ SINGLE TRANSFER. Timing is identical to the LORQ Single Transfer, except that the destination module has set its Busy flip-flop at some earlier time. Busy signals are therefore not involved.

HIRQ MULTIPLE TRANSFER. The HIRQ is kept high until the last SLEN occurs. In addition, the MRQ line is raised, which allows Data to be put on the Bus in successive cycles rather than alternate cycles (as in LORQ Multiple Transfer). The first transmission will occur two cycles after Request (HIRQ) to allow priority to be established (HIINH). Thereafter, transfers are consecutive, and the MDU must supply DES addresses at the times shown.

LORQ/HIRQ TRANSFER. Unlike preceding Multiple Transfers (both LORQ and HIRQ), a LORQ/HIRQ double transfer is addressed to only one Destination. Both LORQ and HIRQ are raised simultaneously, and DES is held constant. The Lo Request is granted first (LOINH), which starts SL at T2, then HIINH occurs and keeps SL set during T3. The first set of data (low priority) is strobed out during T2, and the MDU uses SLEN to switch registers for the second data transmission (high priority) at T3.


BUSES
MODULE 1
MCU
MODULE DATA UNIT
SATELLITE SUPPLY
P.S.
DC
2
3
4
5
6
3-PHASE AC
REGULATED AND UNREGULATED
(7 LINES)
ETC.
TERMINAL BLOCK
REGULATED
UNREGULATED
NEUTRAL
3-PHASE
208/230 VAC
REGULATOR
120V PK
UNREGULATED
REGULATED

POWER DISTRIBUTION

Rather than one large central dc power supply, Omega uses a primary voltage pre-regulator to supply a clipped, regulated 3-phase waveform to "satellite" power supplies associated with each module.

This pre-regulation of the ac input lessens the degree of regulation required of the dc regulators, permitting simpler, less costly circuitry.

Furthermore, each satellite supply needs to generate only those voltages actually required by the associated module.

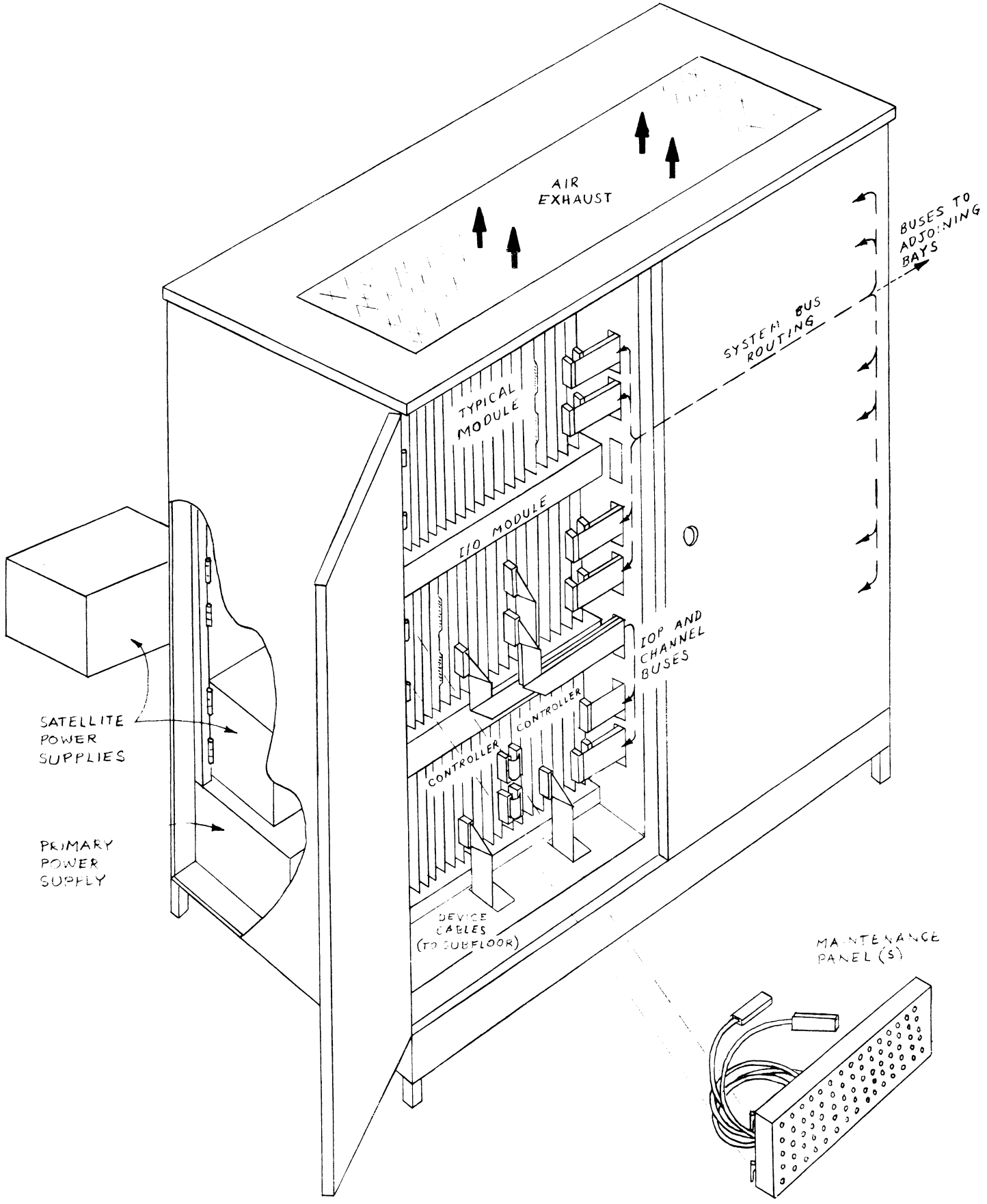
Efficient power distribution is an additional advantage, owing to the use of comparatively high voltage (120V peak). Large conductor buses are not necessary, and conductor losses are kept to a minimum.

The input is an isolated delta configuration for simplicity, with selectable taps for 208V (North American) or 230V (European) operation. Sources of 50 or 60 Hz may be used.

Each phase is separately controlled to minimize mutual interference.

The output to the satellite supplies is a Y (4-wire) configuration, for both the regulated and unregulated lines. (The unregulated supply is primarily for the ventilating fans.)

When loaded to capacity (10A per 208V line, or 8.7A per 230V line), about 3.5 kilowatts is drawn from the source.


AIR EXHAUST
BUSES TO ADJOINING BAYS
SYSTEM BUS ROUTING
TYPICAL MODULE
I/O MODULE
IOP AND CHANNEL BUSES
CONTROLLER
CONTROLLER
SATELLITE POWER SUPPLIES
PRIMARY POWER SUPPLY
DEVICE CABLES (TO SUBFLOOR)
MAINTENANCE PANEL (S)

SYSTEM PACKAGING

The Omega system is packaged in a system cabinet expressly designed for minimum bus length and convenient access to internal hardware.

The system illustrated, not necessarily typical, accommodates 5 modules and 2 in-cabinet device controllers. The controllers are shown in the bottom card cage of the left bay, each occupying half of the card cage. The middle module of the left bay is assumed to be an I/O Module, and the top module might be, for example, a Central Processor Unit. For completeness, the right bay could be assumed to contain 3 memory modules. Larger systems would have additional bays, 3 modules per bay. A full 15-module system would therefore occupy 5 bays.

The satellite power supply for each module is located to the rear of each module, and swings out for access to the power supply circuits and the module backplane. The primary supply is located at the bottom rear of the mainframe, and is capable of supplying regulated ac to those satellites immediately above.

Each module card cage can accept up to 24 printed-circuit cards. The boards are 12 x 12 inches in size, and have a backplane pin capacity of 140 pins. There is a total of 90 microcircuit locations (i.e., 6 x 15 grid). Those cards that have front-edge connection capability normally provide two 50-pin edge connectors.

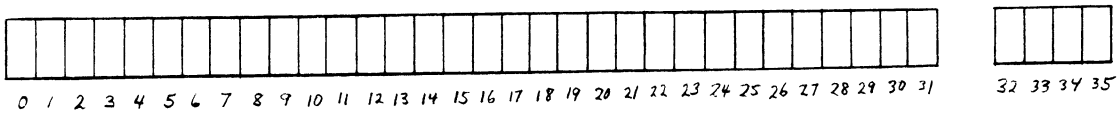
The system bus is routed horizontally in a channel of the mainframe between the top and middle modules, and vertically to each module in a vertical channel on the right side of each bay. So that the bus stubs may be as short as possible, the MCU cards which accept the stubs are always the rightmost two cards of each module. Provision is also made to route other cables through the same channels, such as the IOP (I/O Processor) and high-speed Channel buses, as illustrated. Jumpering of the IOP bus between controllers is shown, between the middle two cards of the controller card cage. Since the Channel bus (coming in to the second card from the right) is not shown jumpered to the left controller, the implication is that the left controller is for a low speed device, and the right controller is for a high speed device. (The descriptions and connection of I/O buses are given later in this manual.)

The bottom front of the cabinet contains a cavity in which connector-type conversions can be made for peripheral device cables. It is assumed that device cabling will be routed under a subfloor of the system site.

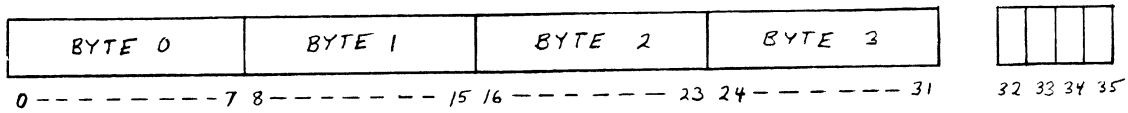
Module ventilation occurs by drawing cool air in at the bottom of the system, and forcing it up through the three modules and exhausting at the top. This provides very efficient cooling for the logic elements. The power supplies have less critical heat rises, and are separately cooled.

A maintenance panel is available for each type of module, and may be moved to test other modules of the same type by means of disconnect hinges. Electrical connection is made by front-edge connectors to the module cards. When doing extensive service, the cabinet doors and panels may be removed completely.

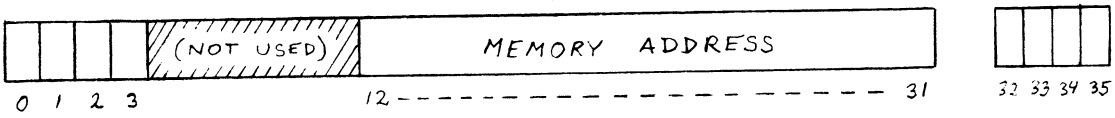
COMPUTER WORD



BYTE FORMAT



MEMORY COMMAND WORD



(20 BITS = $1,048,576_{10}$)

MEMORY OPCODE	MNEMONIC	FUNCTION
0 0 0 0	NOP	No Operation
0 0 0 1		
0 0 1 0	RBWØ	Read Byte/ Write 0
0 0 1 1	CBWØ	Clear Byte/ Write 0
0 1 0 0	RMW	Read/Modify/Write
0 1 0 1	CW	Clear/Write
0 1 1 0	RBW1	Read Byte/Write 1
0 1 1 1	CBW1	Clear Byte/Write 1
1 0 0 0		
1 0 0 1		
1 0 1 0	RBW2	Read Byte/Write 2
1 0 1 1	CBW2	Clear Byte/Write 2
1 1 0 0	RW	Read/Write
1 1 0 1		
1 1 1 0	RBW3	Read Byte/Write 3
1 1 1 1	CBW3	Clear Byte/Write 3

MEMORY WORD CHARACTERISTICS

Omega-32 uses basically a 32-bit word in transmissions between modules. Accompanying each word that goes in or out of memory are 4 parity bits, one for each 8 bits of the basic word (even-parity).

Accordingly, Omega memory modules store information as 36-bit words. Additionally, however, provision is also made to accept, store, and retrieve 8-bit bytes. Thus, information stored in memory may be in either word format or byte format. In byte format, four bytes are packed in one memory location, as shown. Byte instructions will always specify which of the four byte positions is being referenced.

Note that bits are numbered from 0 (most significant) on the left, to 31 (least significant) on the right. Bytes are numbered 0 through 3, left to right. Parity bits correspond numerically: bit 32 for bits 0-7, 33 for 8-15, 34 for 16-23, and 35 for 24-31.

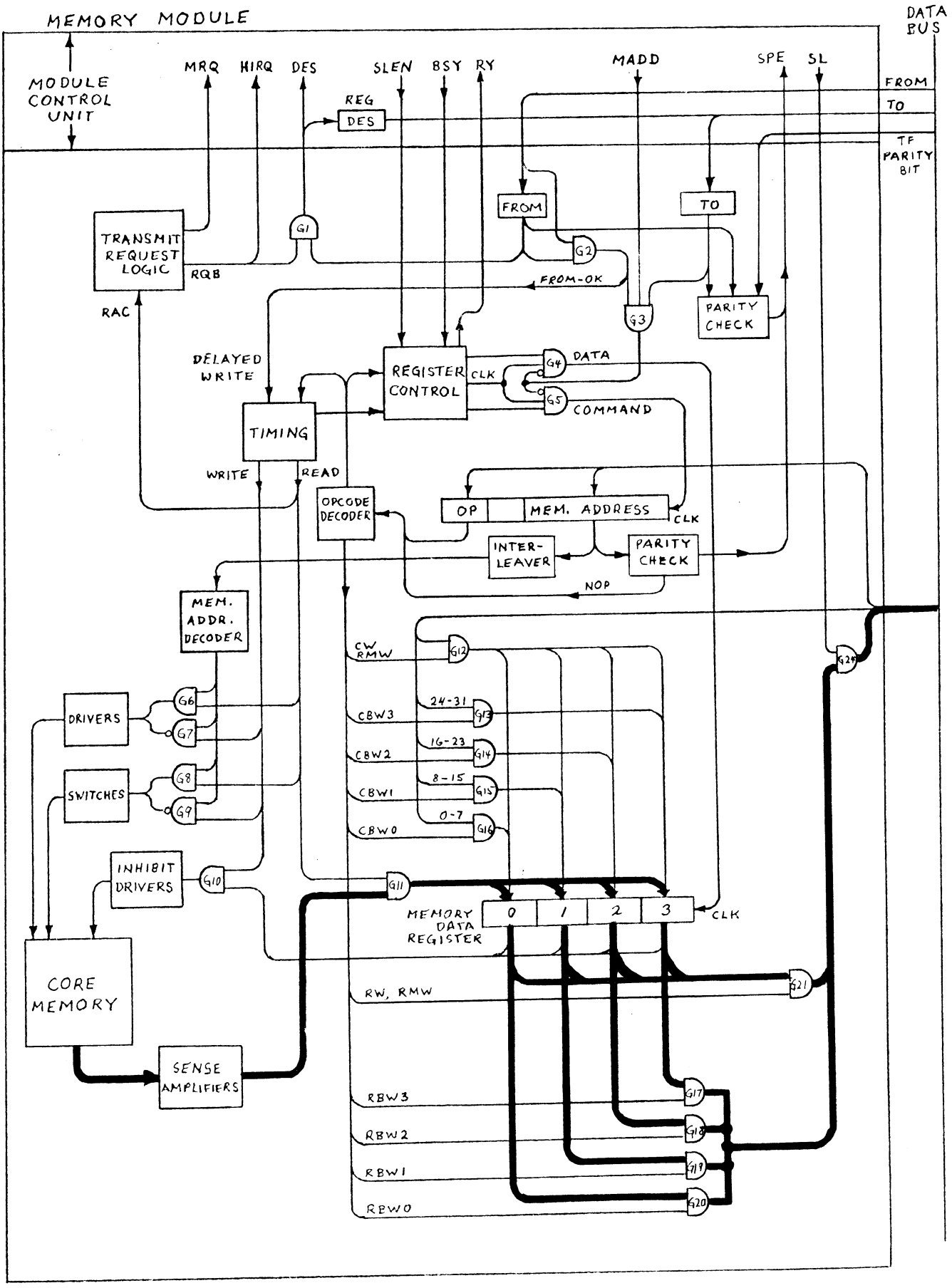
The third word shown at left is the Memory Command Word. All memory transfers consist of at least two transmissions. The first transmission must be the Command Word, directed to the memory module, specifying the memory address affected, plus an Opcode specifying the function to be performed. The second transmission transfers data in or out of memory on the Data Bus, according to the Opcode command.

The table at left lists all Memory Opcodes. (Those not used are interpreted as NOP's.) These will be further explained in the following pages on Memory; but, in brief, the Opcodes provide 5 basic operations, as follows:

RW	Read/Write	(to retrieve a 32-bit word)
CW	Clear/Write	(to store a 32-bit word)
RMW	Read/Modify/Write	(to read a word, modify it in the initiating module, and store it back in the same memory location)
RBW	Read Byte/Write	(to retrieve one of Bytes 0-3)
CBW	Clear Byte/Write	(to store in one of Bytes 0-3)

The memory module does not check parity on the information that it stores and reads out. It does, however, check parity on the Memory Command Word (does a NOP if an error is detected) and on the eight TO/FROM bits (pulls on the System Parity Error line). The parity of memory data must be checked by the receiving module when the information is read out. The receiver therefore, in a single check, simultaneously checks for possible errors in the earlier transmission to memory, the storage in memory, and the transmission from memory.

The following two pages describe the logic operations involved in reading and writing memory data. The descriptions apply to all memory modules, regardless of memory speed (750 nsec, 950 nsec, 2 μsec, etc.)


MEMORY MODULE
DATA BUS
MODULE CONTROL UNIT
MRQ
HIRQ
DES
SLEN
BSY
RY
MADD
SPE
SL
FROM
TO
TF PARITY BIT
REG DES
TRANSMIT REQUEST LOGIC
RQB
RAC
G1
G2
G3
FROM
TO
FROM-OK
PARITY CHECK
DELAYED WRITE
REGISTER CONTROL
CLK
G4
DATA
G5
COMMAND
TIMING
WRITE
READ
OPCODE DECODER
OP
MEM. ADDRESS
CLK
INTER-LEAVER
PARITY CHECK
NOP
MEM. ADDR. DECODER
CW RMW
G12
G24
DRIVERS
G6
G7
CBW3
24-31
G13
CBW2
16-23
G14
SWITCHES
G8
G9
CBW1
8-15
G15
CBW0
0-7
G16
INHIBIT DRIVERS
G10
G11
MEMORY DATA REGISTER
0
1
2
3
CLK
CORE MEMORY
RW, RMW
G21
SENSE AMPLIFIERS
RBW3
G17
RBW2
G18
RBW1
G19
RBW0
G20

READ MEMORY

LOAD COMMAND WORD

1. The initial state of Register Control is the COMMAND position (G5 enabled).

2. The Command Word from the initiating module is clocked into the Memory Address Register from the Bus. The four TO (in) bits compare with MADD ("this module") at G3, which locks the word in the register. Register Control goes not-ready (RY), and read timing starts (regardless of whether a read action will be completed by the particular command).

3. Bits 12-31 are decoded and apply enabling signals to the Driver and Switch circuits. The X-Y co-ordinate outputs will select one specific memory cell when activated. If the system uses memory interleaving, the appropriate high and low order bits are interchanged by the Interleaver wiring. (An example of memory interleaving could be to have a CPU or I/O Module address different memory modules for odd and even addresses, or perhaps four different modules determined by the two least significant bits of the address. It is the responsibility of the CPU's or IOM's Address Mapper to select the correct module, and it is the responsibility of the memory module's Interleaver to arrange the address bits for addressing its own core without gaps. The net effect is that the CPU or IOM can make successive memory transfers a faster rate, since it can be transmitting to an alternate module while waiting for another to complete its cycle.)

4. The decoded Memory Opcode (Bits 0-4) immediately switches Register Control to the DATA position (G4), and initiates one of the following actions (this page and the next).

READ WORD OR BYTE

1. Reading a word (RW) or a byte (RBW) begins with Read Timing signals being sent to Core Memory via G6 and G8. (The Memory Data Register is also set to all-ones at this time.)

2. All cores in the selected location are switched to the "one" state, so that any cores that were zero will change state. This change of state is detected by the Sense Amplifiers, and resets the corresponding flip-flop in the Data Register when strobed by a signal to G11.

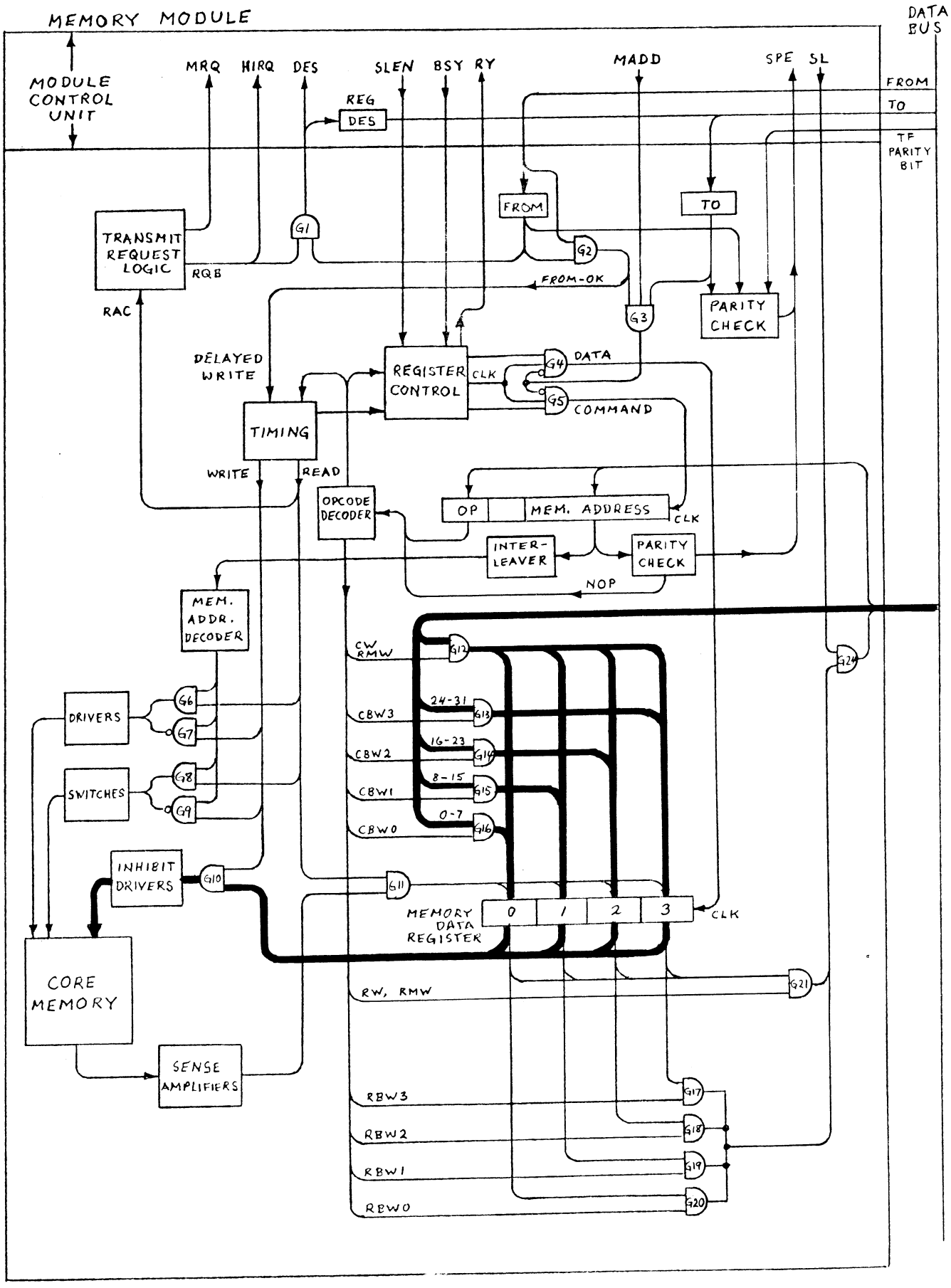
3. A signal called RAC (Read Almost Complete), which is delayed from Read by a fixed time, causes the Transmit Request Logic to issue a High-Priority Request (HIRQ). This permits priority checks in MCU to proceed concurrently.

4. When data is in the register, an appropriate strobe signal to either G21 (for word reading) or one of G17 through G20 (for byte reading) applies the data to G24. (Note that bytes are always routed on Bus lines 24-31, regardless of original position.)

5. When selected, G24 sends data to the initiating module via the Data Bus, along with the FROM and TO codes. The TO code is derived via G1 from the stored FROM code, which originally came with the Command Word.

6. The word in the Data Register is now written back into memory through the Inhibit Drivers (see Write, step 4).

7. The Ready signal switches the Register Control Logic back to the COMMAND position for the next Command Word, and clears the Busy flip-flop in the MCU.


MEMORY MODULE
DATA BUS
MODULE CONTROL UNIT
MRQ
HIRQ
DES
SLEN
BSY
RY
MADD
SPE
SL
FROM
TO
TF PARITY BIT
REG DES
TRANSMIT REQUEST LOGIC
RQB
RAC
G1
G2
G3
FROM
TO
FROM-OK
PARITY CHECK
DELAYED WRITE
REGISTER CONTROL
CLK
G4
DATA
G5
COMMAND
TIMING
WRITE
READ
OPCODE DECODER
OP
MEM. ADDRESS
CLK
INTER-LEAVER
PARITY CHECK
NOP
MEM. ADDR. DECODER
CW RMW
G12
G24
DRIVERS
G6
G7
CBW3
24-31
G13
CBW2
16-23
G14
SWITCHES
G8
G9
CBW1
8-15
G15
CBW0
0-7
G16
INHIBIT DRIVERS
G10
G11
MEMORY DATA REGISTER
0
1
2
3
CLK
CORE MEMORY
RW, RMW
G21
SENSE AMPLIFIERS
RBW3
RBW2
RBW1
RBW0
G17
G18
G19
G20

WRITE, AND READ/MODIFY/WRITE

LOAD COMMAND WORD

Same as on preceding page.

WRITE WORD OR BYTE

1. Writing a word (CW) or a byte (CBW) begins with a simulated read operation, as described on the preceding page, except that previous stored data is cleared instead of being read-out. This occurs simply by not enabling G11. (Or, for byte operations, G11 is selectively enabled to load in the Register only the 3 bytes not involved in the current operation.) Write is suspended until the initiating module supplies data (next step).

2. The Register Control switches to DATA immediately upon starting the Clear operation. The module is now ready to receive data. However, comparator G2 will act only on data that has the correct FROM code. The resulting FROM-OK signal initiates a Delayed Write operation.

3. Incoming data from the Bus is clocked into the Memory Data Register via G12 (for word writing) or one of G13 through G16 (for byte writing). If the input is a byte, it will be received from the bus in the desired byte position.

4. Write Timing signals strobe G10 and G7/9 to write the data into memory through the Inhibit Drivers, at the location selected by the Switches and Drivers. The core-magnetizing direction is the reverse of Read (now tending to write zeros), so any Register bits that are "one" will inhibit writing of a zero.

5. The Ready signal switches Register Logic back to the COMMAND state.

READ/MODIFY/WRITE

This operation, which combines both the Read and Write operations, is for words only. The Write operation does not proceed until data is returned, so the initiating module has the opportunity to modify the read-out word before returning it for the write operation. The following sequence is condensed.

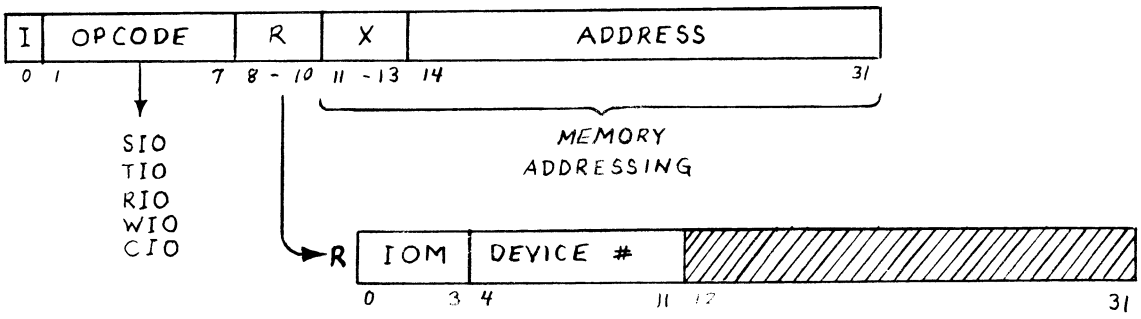
1. Read Timing signals strobe G6/8, causing the Drivers and Switches to read Core data out to the Sense Amplifiers.

2. The Sense Amplifier output is strobed into the Memory Data Register at G11, then out to the Bus via G21 and (when selected) G24. Register Control switches to DATA.

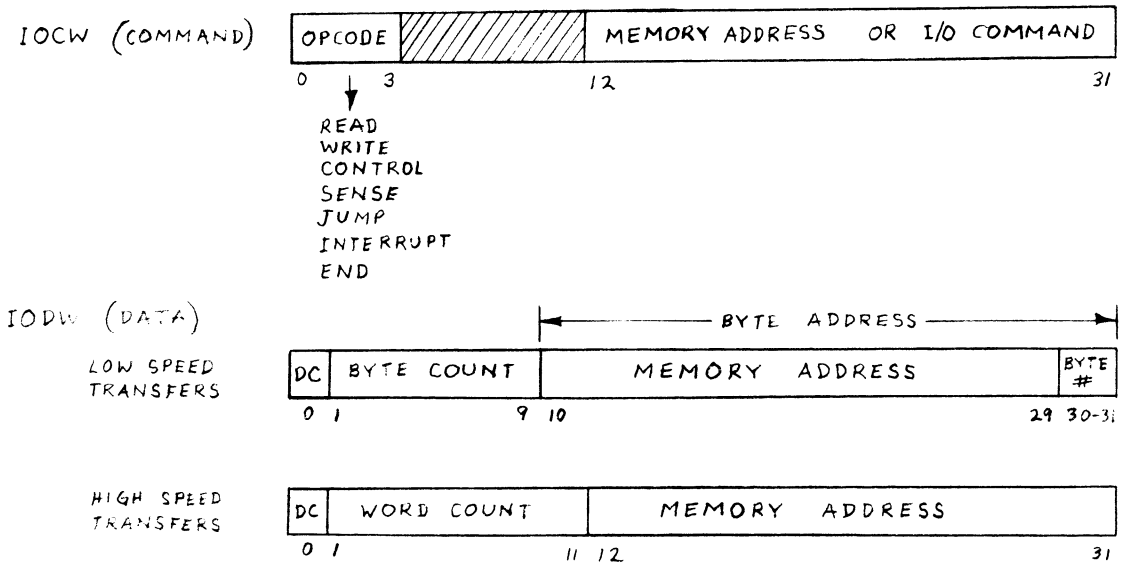
3. The initiating module receives the word, modifies it as desired, then re-transmits it.

4. The returned word is strobed into the Memory Data Register at G12, and the accompanying FROM code initiates a Delayed Write. Write Timing signals strobe G10 and G7/9, writing the word back into memory.

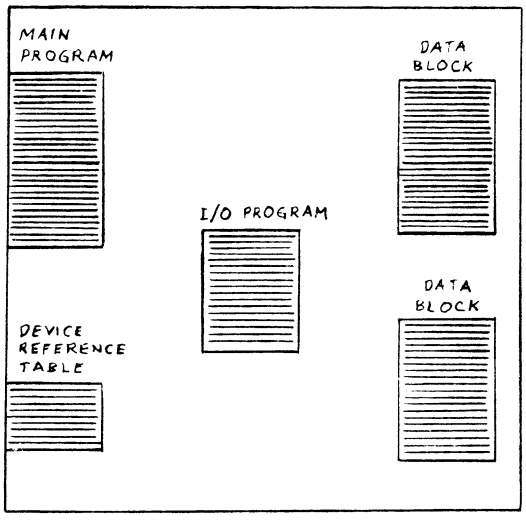
FORMAT OF I/O INSTRUCTIONS



FORMATS OF I/O PROGRAM WORDS



MEMORY MAP (I/O)



OMEGA I/O PROGRAMMING

The Omega system architecture provides for 5 input/output instructions and addressing capability for 256 devices. The 5 I/O instructions are: Start I/O, Test I/O, Read I/O, Write I/O, and Command I/O. The nominal limit of 256 devices assumes only one I/O Module in the system. More devices can be accommodated by adding an additional I/O module; however, special programming considerations must also be made since the Device Reference Table (discussed later) provides entries for a maximum of 256 devices.

Assuming that there is an I/O Program stored in memory for each device, I/O transfers are initiated simply by addressing a Start I/O instruction to the desired device. The program in progress may continue immediately to succeeding instructions while the I/O Module independently executes the I/O transfer. When the transfer is complete, (typically) the I/O Module interrupts a CPU module to signify completion.

Other transfer modes are also possible. For example, the I/O Program may be bypassed for CPU-I/O transfers by using the direct write/read instructions (WIO, RIO).

Command I/O instruction CIO permits a command to be directed to the device, without a data transfer being involved. Test I/O, TIO, permits status checking of the device.

Definitions of the 5 input/output instructions are as follows.

SIO R,M (Start I/O)

The left 12 bits of register R contain the I/O Module number (4 bits) and the device number (8 bits). The right 20 bits of register R contains the address of an I/O program which is to be executed by the I/O processor. If rejected, a branch to M occurs. Condition Code is unaffected. SIO is a privileged instruction.

TIO R,M (Test I/O)

The left 12 bits of register R contain the I/O Module number (4 bits) and the device number (8 bits). A halfword containing the status of the device is returned to the right half of M. Condition Code is unaffected. TIO is a privileged instruction.

RIO R,M (Read I/O)

The left 12 bits of register R contain the I/O Module number (4 bits) and the device number (8 bits). If successful, a data inbound halfword is read directly from this device into the right half of R. The left half of R is undisturbed. Processing does not continue until the read is completed. If rejected, a branch to M occurs. Condition Code is unaffected. RIO is a privileged instruction.

WIO R,M (Write I/O)

The left 12 bits of register R contain the I/O Module number (4 bits) and the device number (8 bits). A halfword is written directly from the right halfword of R into the device data outbound halfword. Processing does not continue until the write is completed. If rejected, a branch to M occurs. Condition Code is unaffected. WIO is a privileged instruction.

CIO R,M (Command I/O)

The left 12 bits of register R contain the I/O Module number (4 bits) and the device number (8 bits). A halfword is written directly from the right halfword of M into the device command register. Processing continues immediately. This command can also be used to send a halfword to another CPU or to any other module whose number is placed in register R. Condition Code is unaffected. CIO is a privileged instruction.

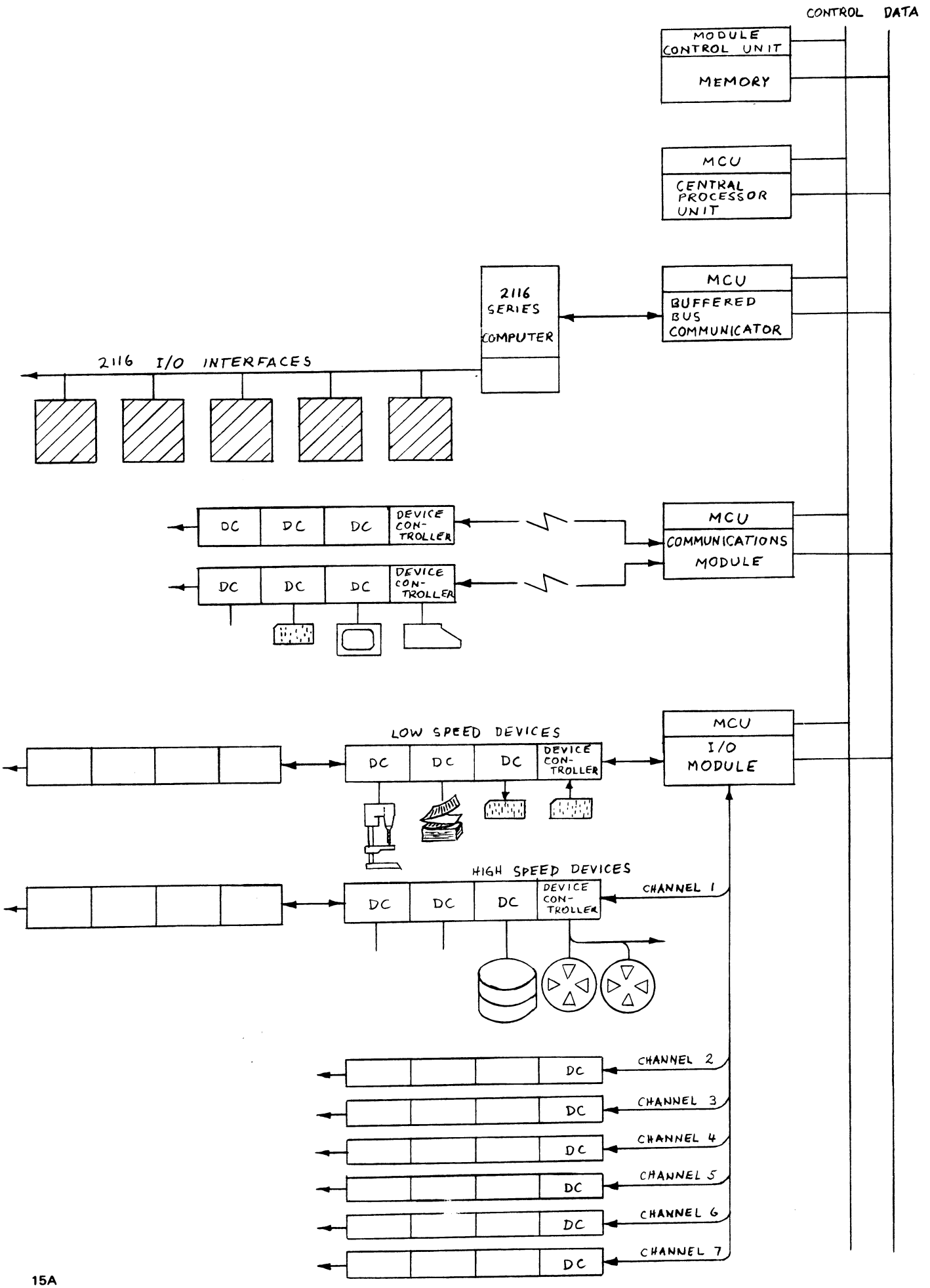
The format of I/O instructions is shown in the upper figure on the facing page. Bits 1-7 specify the instruction, bits 8-10 point to one of the R registers in the CPU, and bits 11-31 point to the starting address, in memory, where the I/O program for this device is stored. The R register identified by bits 8-10 will previously have been loaded, as shown, by the program in progress (hereinafter called "Main Program"). Bits 0-3 of R give the I/O Module number, bits 4-11 give the device number; bits 12-31 are not used.

The I/O program for each device consists of I/O Command Words (IOCW) and I/O Data Words (IODW). If the IOCW is a Read or Write, the Address field is not used and the following word will be an IODW, telling the I/O Module how much data to transfer, and where in memory to start the transfer. If the IOCW is a Control command, bits 16-31 convey control information; if more control information is needed, another IOCW can follow. If the IOCW is a Sense or Jump, bits 12-31 are used to specify a memory location. For Sense, that location receives the 16 Status bits of the device. For Jump, the I/O program simply jumps to that location for its next IOCW. If the IOCW is an Interrupt, the I/O Module requests an interrupt of a CPU. End terminates the I/O program.

There are two different types of I/O Data Words, one for low speed transfers, and one for high speed transfers. For low speed transfers, data is transferred between I/O and memory one byte at a time. The Count value is therefore a byte count, and the byte number is shifted into the two least significant bits of the word, so that for programming purposes bits 10-31 can be thought of as a "byte address". For high speed transfers, data bytes are packed into full words in the I/O Module, so that data is transferred between I/O and memory one word at a time. The Count value is therefore, for convenience, a word count.

In the case of high-speed transfers, hardware provision is made for a device to "inhibit decrement" for a given number of counts, provided it has the counting logic to perform this function. Thus the Count can then become a record count, rather than a word count.

To visualize where all these different word-types are stored, refer to the Memory Map. (Memory addresses are assumed to ascend from the lower left corner.) The I/O instructions are imbedded in the Main Program. The IOCWs and IODWs make up the I/O Program. The Data Blocks (normally in a different Memory Module from programs) are locations where data is transferred in or out. The Device Reference Table contains a doubleword entry for each of the 256 devices (512 locations). The entry in this table is initialized at the start of each transfer (first action of the SIO instruction). This allows several devices to use the same I/O program, or conversely, one device may use any of a number of different I/O programs. The Device Reference Table occupies dedicated locations 1000-1777 octal.


CONTROL
DATA
MODULE CONTROL UNIT
MEMORY
MCU
CENTRAL PROCESSOR UNIT
2116 SERIES COMPUTER
MCU
BUFFERED BUS COMMUNICATOR
2116 I/O INTERFACES
DC
DC
DC
DEVICE CON-TROLLER
DC
DC
DC
DEVICE CON-TROLLER
MCU
COMMUNICATIONS MODULE
LOW SPEED DEVICES
DC
DC
DC
DEVICE CON-TROLLER
MCU
I/O MODULE
HIGH SPEED DEVICES
DC
DC
DC
DEVICE CON-TROLLER
CHANNEL 1
CHANNEL 2
DC
CHANNEL 3
DC
CHANNEL 4
DC
CHANNEL 5
DC
CHANNEL 6
DC
CHANNEL 7
DC

THE OMEGA I/O SYSTEM

There are several types of modules that communicate with I/O devices. Three are shown in the illustration at left: the Buffered Bus Communicator, the Communications Module, and the I/O Module.

The Buffered Bus Communicator provides bus access for special purpose modules. The BBC itself provides generalized communication with the system buses; special purpose interfaces are additionally required in order to adapt the BBC logic to specific applications. A BBC interface exists for 2116-family computers, allowing such computers to look like a module in the Omega system.

The Communications Module provides connection for remote terminals. Since the Communications Module uses a different Device Reference Table than that of the I/O Module, there may be 256 terminals in addition to the 256 devices connected to the I/O Module.

The I/O Module provides for two different types of transfers: low-speed, under control of a microprocessor within the module, and high-speed, bypassing the microprocessor through one of 7 Direct Memory Access (DMA) channels. The DMA channels are initialized by the microprocessor, then perform their transfers in blocks. Once a device gets the channel, no other device on that channel may begin until the current device finishes its block transfer. Data from low-speed devices, however, is multiplexed byte by byte on a priority basis.

The Device Controllers translate the general I/O Module commands into specific timing and control required for the peripheral devices. Several Device Controllers can be installed in a Controller rack (card cage), and the I/O buses are cabled in series through each rack. Priority order is from the nearest Controller on the bus (nearest the I/O Module) to the farthest (lowest priority).

The descriptions on the following pages explain the operation of the I/O Module, and for simplicity will show only 1 of the 7 DMA Channels, with only one Device Controller of each type (high speed and low speed). Note, however, that each Channel can handle several Device Controllers, and some Controllers may handle several devices of a given type.

The organization of the following descriptions provides sequencing information first (pages 16 through 22), showing the information exchanges that occur to accomplish both low and high speed transfers. This is followed by logic descriptions (pages 23 through 33), discussing the internal logic of each sub-unit of the I/O Module, with little relation to time sequences.

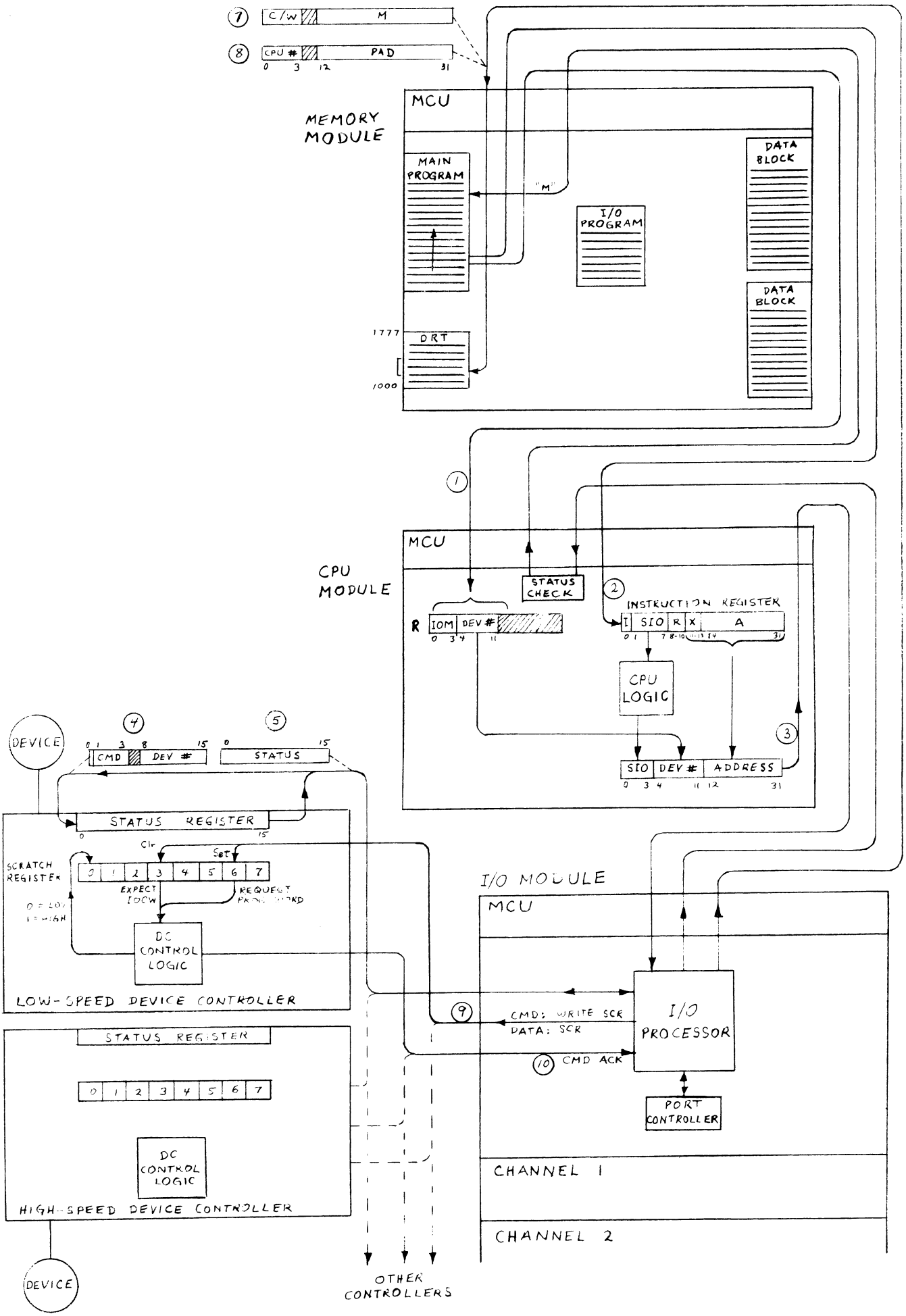
For now, the following definitions of the I/O Module sub-units will suffice.

MODULE CONTROL UNIT. Receives data from the other Omega modules, and determines when this module (the I/O Module) may transmit its data to the other modules, based on its assigned priority.

I/O PROCESSOR. Multiplexes low-speed transfers, and initializes DMA Channels for high-speed transfers.

PORT CONTROLLER. Selects which one of 8 "ports" (7 DMA Channels, plus the I/O Processor) may make a request to the MCU to transmit.

DMA CHANNEL. The high-speed transfer logic.


7
C/W
M
8
CPU #
PAD
0 3 12 31
MEMORY MODULE
MCU
MAIN PROGRAM
"M"
I/O PROGRAM
DATA BLOCK
DATA BLOCK
1777
DRT
1000
1
CPU MODULE
MCU
STATUS CHECK
2
INSTRUCTION REGISTER
R
IOM
DEV #
I
SIO
R
X
A
CPU LOGIC
3
SIO
DEV #
ADDRESS
DEVICE
4
CMD
DEV #
5
STATUS
STATUS REGISTER
SCRATCH REGISTER
Clr
Set
0 1 2 3 4 5 6 7
EXPECT IOCW
REQUEST
0 = LOW
1 = HIGH
DC CONTROL LOGIC
LOW-SPEED DEVICE CONTROLLER
I/O MODULE
MCU
9
CMD: WRITE SCR
DATA: SCR
I/O PROCESSOR
10
CMD ACK
PORT CONTROLLER
CHANNEL 1
CHANNEL 2
STATUS REGISTER
0 1 2 3 4 5 6 7
DC CONTROL LOGIC
HIGH-SPEED DEVICE CONTROLLER
DEVICE
OTHER CONTROLLERS

STARTING AN I/O TRANSFER

As stated earlier, I/O transfers are initiated by addressing a "Start I/O" (SIO) instruction to the desired device. But before the I/O Module can process the SIO, it must be told where its I/O Program is. The location of the I/O Program can be changed at any time, and in fact, a choice of several programs may be available, depending on the application.

The diagram at left shows how the Main Program establishes the Program Address (PAD), and gets the I/O program started. The I/O Program itself will tell the device what to do: start, stop, input data, output data, how much data to transfer, where in memory to put or take the data, and so on. These actions of the I/O Program will be discussed on succeeding pages. Operations are identical for both low and high speed devices until actual data transfer begins (page 19).

The Program Address is established by the following procedure. (Step numbers refer directly to numbers on the facing diagram.)

START SIO SEQUENCE

1. Main Program loads Left Half of R (i.e., one of the 8 "R" Registers in the CPU), to identify the desired device.

2. Main Program issues SIO instruction to the CPU.

3. CPU relays the SIO (now including the Device Number from R, and the computed address of the I/O Program from the SIO instruction) to the I/O Module. Within the I/O Module, the I/O Processor interprets the SIO and begins to execute an SIO program sequence from its internal Read-Only-Memory (ROM). The first action of the ROM program is to obtain the status of the device (next 3 steps).

CHECK STATUS

4. The I/O Processor sends a command word to the Device Controller, consisting of the Command code (CMD) for "Read Status", and 8 bits of Device address (DEV #). The complete set of CMD codes (ground-true, positive-false) is as follows:

CMD Bit 1	2	3	Command
0	0	0	NOP
0	0	1	Read Data from Device
0	1	0	Read Status from Device
0	1	1	Write Data to Device
1	0	0	Write Command to Device
1	0	1	Write Scratch to Device
1	1	0	Issue DIL (Data Index Low) to Channel
1	1	1	Issue DIH (Data Index High) to Channel

5. Controller sends 16 bits of Status, with CMD ACK (Command Acknowledge) to I/O Processor, via Data-In lines.

6. I/O Processor transmits Status to right half of a register in the CPU. If bit 29 says that the device is not ready to accept the SIO, a branch to M occurs; otherwise, the SIO procedure continues, and the CPU is from now on not involved.

INITIALIZE DRT

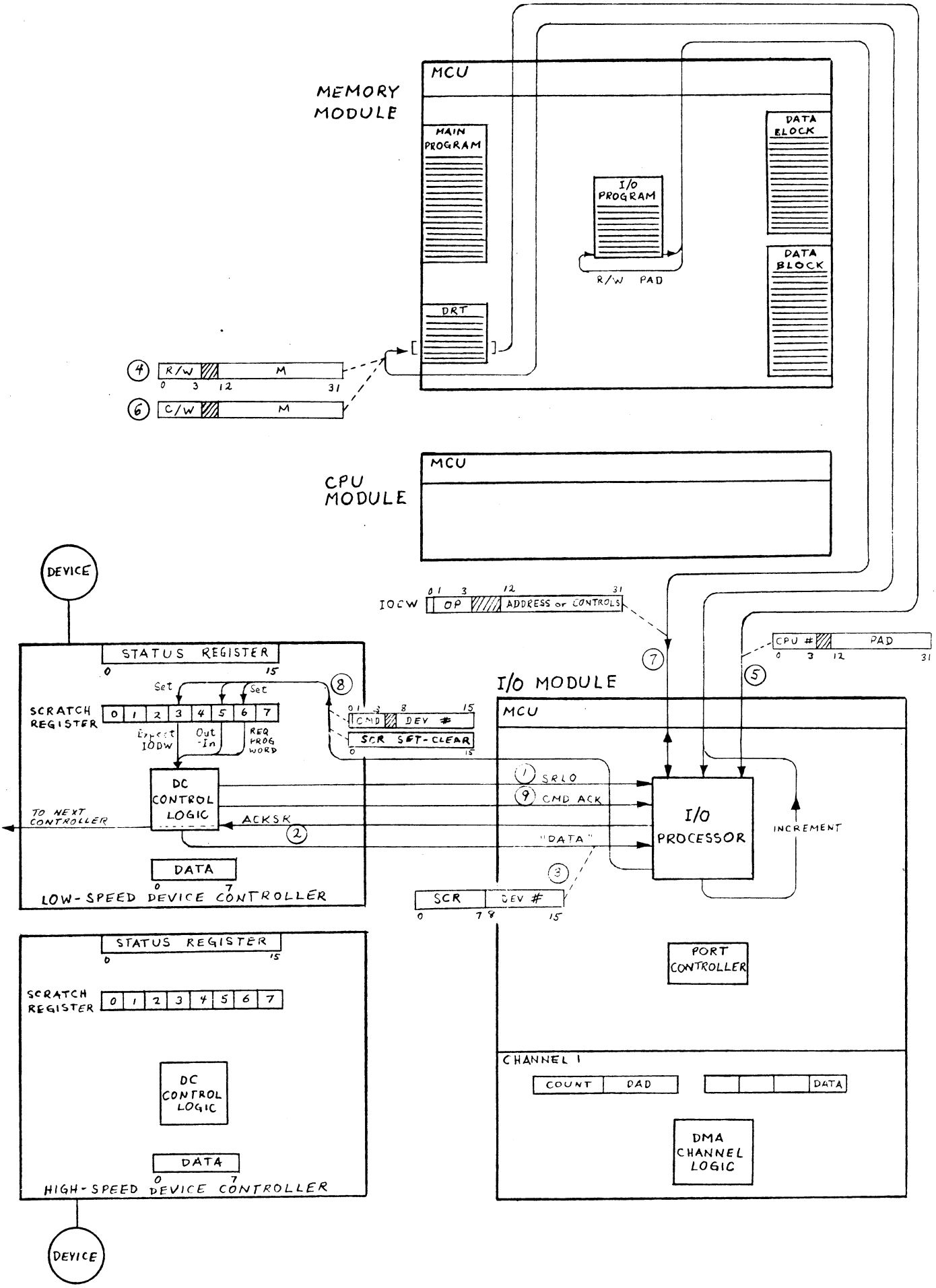
7. I/O Processor addresses first location of Device's 2-word allocation in the Device Reference Table (DRT), with a Clear/Write memory command.

8. I/O Processor transmits Program Address PAD and CPU Module number to DRT.

START I/O PROGRAM

9. I/O Processor tells Controller to request a Program Word. This is done by sending a "Write Scratch" command word (see step 4 above) to the Controller and placing the desired loading of the Scratch Register (SCR) on the Data-Out lines. When the Scratch word is written into the Scratch Register, bit 6 (set) tells the Controller to request a Program Word, and bit 3 (cleared) tells the Controller to expect an I/O Control Word (IOCW). Bit 0 is permanently wired set or cleared, depending on whether the device is low or high speed.

10. Command Acknowledge CMD ACK frees the I/O Processor. I/O Processor may now process any other operation phase for any other device, while waiting for the Controller to request its Program Word (next page)


MEMORY MODULE
MCU
MAIN PROGRAM
I/O PROGRAM
DATA BLOCK
DATA BLOCK
R/W PAD
DRT
4
R/W
M
0 3 12 31
6
C/W
M
CPU MODULE
MCU
DEVICE
IOCW
0 1 3 12 31
OP
ADDRESS or CONTROLS
7
CPU # PAD
0 3 12 31
5
I/O MODULE
MCU
STATUS REGISTER
0 15
8
Set
Set
SCRATCH REGISTER
0 1 2 3 4 5 6 7
Expect IODW
Out -In
REQ PROG WORD
0 1 3 8 15
CMD DEV #
SCR SET-CLEAR
0 15
1 SRLO
9 CMD ACK
TO NEXT CONTROLLER
DC CONTROL LOGIC
ACKSR
2
"DATA"
I/O PROCESSOR
INCREMENT
DATA
0 7
LOW-SPEED DEVICE CONTROLLER
3
SCR
DEV #
0 7 8 15
PORT CONTROLLER
STATUS REGISTER
0 15
SCRATCH REGISTER
0 1 2 3 4 5 6 7
CHANNEL 1
COUNT
DAD
DATA
DC CONTROL LOGIC
DMA CHANNEL LOGIC
DATA
0 7
HIGH-SPEED DEVICE CONTROLLER
DEVICE

EXECUTING THE I/O PROGRAM

DEVICE ASKS FOR PROGRAM WORD

1. Device Controller issues SRLO (Service Request Low) to I/O Processor, in response to I/O Processor's command (preceding page).

2. I/O Processor polls all Controllers, in "series", with ACKSR (Acknowledge Service Request) to determine whose SRLO it has.

3. Highest-priority requesting device responds by reading out its Device Number and Scratch Register contents. Remember that bit 6 says that a Program Word is requested.

INCREMENT PAD

4. Before obtaining the Program Word, the I/O Processor first reads PAD out of the DRT (Read/Write memory command).

5. I/O Processor receives PAD and increments it.

6. The incremented value of PAD is stored back in DRT (Clear/Write memory command).

EXECUTE IOCW

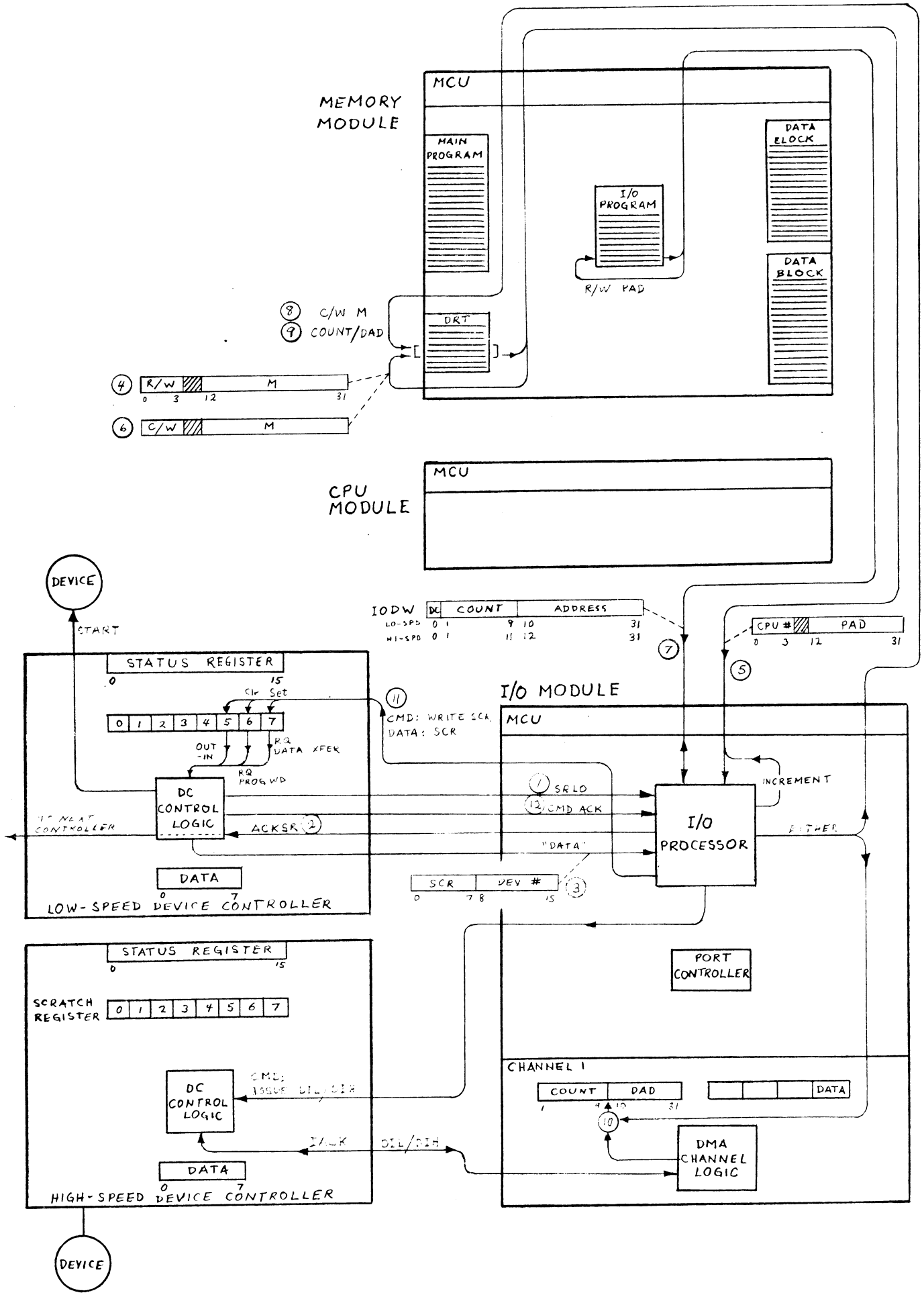
7. I/O Processor reads out the contents of PAD (pre-incremented value), which in this case (first Program Word) is an I/O Command Word. The coding of the Opcode bits (1-3) will direct the I/O Processor to one of 7 command programs in ROM, as follows:

IOCW Bit 1	2	3	Command	Definition
0	0	0	END	Terminates I/O Program.
0	0	1	SENSE	16 bits of Status are transmitted to memory (at Address specified in IOCW).
0	1	0	CONTROL	Right 16 bits of IOCW are transferred to Device Command Register (see also page 22).
0	1	1	JUMP	Fetch next IOCW from Address specified in IOCW.
1	0	0	WRITE	Begin output transfer, according to block-size and starting address specified in the IODW immediately following.
1	0	1	READ	Begin input transfer, according to block-size and starting address specified in the IODW immediately following.
1	1	0	INTERRUPT	Device is told to request a CPU Interrupt (SRLO, with Status transmission to CPU).

ASK FOR NEXT WORD

8. I/O Processor tells Controller to ask for next Program Word (by setting bit 6 of SCR) and to expect, for example, an I/O Data Word (IODW) by setting bit 3. Any number of IOCW's could follow the first one, unless the command is READ or WRITE, in which case an IODW must immediately follow.

9. CMD ACK frees I/O Processor.


MEMORY MODULE
MCU
MAIN PROGRAM
I/O PROGRAM
DATA BLOCK
DATA BLOCK
R/W PAD
8 C/W M
9 COUNT/DAD
DRT
4 R/W M
0 3 12 31
6 C/W M
CPU MODULE
MCU
DEVICE
START
IODW DC COUNT ADDRESS
LO-SPD 0 1 9 10 31
HI-SPD 0 1 11 12 31
7
CPU # PAD
0 3 12 31
5
STATUS REGISTER
0 15
Clr Set
0 1 2 3 4 5 6 7
OUT -IN
RQ DATA XFER
RQ PROG WD
11
CMD: WRITE SCR
DATA: SCR
I/O MODULE
MCU
DC CONTROL LOGIC
ACKSR 2
TO NEXT CONTROLLER
1 SRLO
12 CMD ACK
"DATA"
INCREMENT
I/O PROCESSOR
EITHER
DATA
0 7
LOW-SPEED DEVICE CONTROLLER
SCR DEV #
0 7 8 15
3
PORT CONTROLLER
STATUS REGISTER
0 15
SCRATCH REGISTER
0 1 2 3 4 5 6 7
DC CONTROL LOGIC
CMD: ISSUE DIL/DIH
ACK
DIL/DIH
CHANNEL 1
COUNT DAD
1 9 10 31
DATA
10
DMA CHANNEL LOGIC
DATA
0 7
HIGH-SPEED DEVICE CONTROLLER
DEVICE

ESTABLISHING TRANSFER PARAMETERS

DEVICE ASKS FOR PROGRAM WORD

1. Device issues SRLO to I/O Processor, in response to I/O Processor's command.

2. I/O Processor polls all Controllers with ACKSR.

3. Highest-priority requesting device responds by reading out its Device Number and Scratch Register contents (wants a Program Word).

INCREMENT PAD

4. I/O Processor addresses first word of Device's entry in DRT to get PAD.

5. Memory sends PAD.

6. I/O Processor increments PAD, and stores it back in DRT.

GET IODW

7. I/O Processor addresses memory (Read/Write at PAD address), and memory sends its next Program Word (IODW).

then either:

STORE IN DRT

8. For low-speed transfers, I/O Processor addresses second word of Device's allocation in DRT.

9. Sends IODW values (Count and Address) to DRT. Address is also referred to as Data Address, or DAD.

OR STORE IN DMA CHANNEL

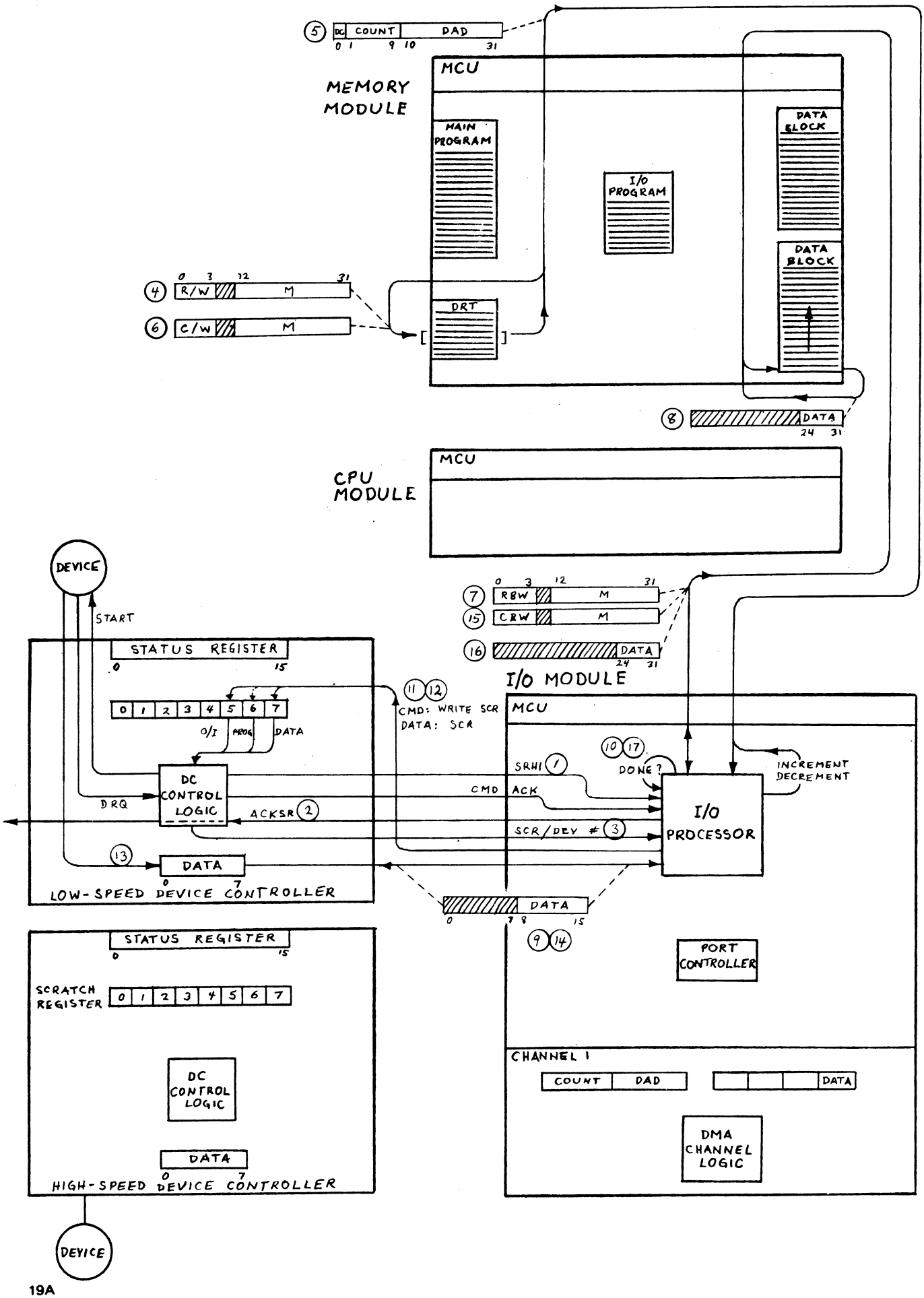
10. For high-speed transfers, I/O Processor tells the Controller (by Commands discussed in step 4 of page 16) to issue DIL and DIH signals to the Channel logic. DIL loads the lower half of IODW in the COUNT/DAD ("Index") register. IACK (Index Acknowledge) allows DIH to load the lower half of IODW, and a Channel-Busy signal (not shown) inhibits other devices on the same channel.

START DEVICE

11. I/O Processor tells Controller to ask for a Data transfer (set bit 7) and to clear the Program Word request (bit 6). Bit 7 will start device (e.g., Read, or some mechanical operation), and will enable Channel Interface, if a high-speed transfer.

12. CMD ACK frees I/O Processor.

To page 19 for low-speed devices.
To page 20 for high-speed devices.


5
DC
COUNT
DAD
0 1 9 10 31
MEMORY MODULE
MCU
MAIN PROGRAM
I/O PROGRAM
DATA BLOCK
DATA BLOCK
DRT
4
R/W
M
0 3 12 31
6
C/W
M
8
DATA
24 31
CPU MODULE
MCU
DEVICE
START
STATUS REGISTER
0 15
0 1 2 3 4 5 6 7
O/I
PROG
DATA
DC CONTROL LOGIC
DRQ
ACKSR 2
13
DATA
0 7
LOW-SPEED DEVICE CONTROLLER
7
R8W
M
0 3 12 31
15
C8W
M
16
DATA
24 31
I/O MODULE
11 12
CMD: WRITE SCR
DATA: SCR
MCU
10 17
DONE?
INCREMENT
DECREMENT
SRHI 1
CMD
ACK
SCR/DEV # 3
I/O PROCESSOR
DATA
0 7 8 15
9 14
PORT CONTROLLER
STATUS REGISTER
0 15
SCRATCH REGISTER
0 1 2 3 4 5 6 7
DC CONTROL LOGIC
DATA
0 7
HIGH-SPEED DEVICE CONTROLLER
CHANNEL 1
COUNT
DAD
DATA
DMA CHANNEL LOGIC
DEVICE

LOW SPEED TRANSFERS

DEVICE ASKS FOR DATA TRANSFER

1. When ready to read or write (Data Request signal DRQ from Device), Controller issues SRHI (Service Request High) to I/O Processor, in response to I/O Processor's command.

2. I/O Processor polls all Controllers will ACKSR.

3. Highest-priority requesting device responds by reading out its Device Number and Scratch Register contents (wants Data transfer).

UPDATE COUNT AND DAD

4. I/O Processor sends out to DRT (second word) for COUNT/DAD.

5. Memory sends it.

6. I/O Processor increments DAD, decrements COUNT, and returns word to memory.

OUT (DEVICE BOUND)

7. I/O Processor sends to memory for data byte.

8. Memory sends it. (On least significant bit lines of the bus.)

9. I/O Processor transfers data to Controller's Data buffer. This is done by issuing a Command word for "Write Data" to Device (see step 4 of page 16), and placing the data on bit lines 8-15 of the Data-Out bus. Controller returns CMD ACK.

10. I/O Processor checks for Count-zero.

11. If not zero, I/O Processor tells Controller to ask for another Data transfer. CMD ACK frees I/O Processor. Controller re-issues START.

REPEAT TO STEP 1 ABOVE

12. If zero, I/O Processor tells Controller to ask for a new Program Word, and to clear the Data request. CMD ACK frees I/O Processor.

REPEAT TO PAGE 17

IN (MEMORY BOUND)

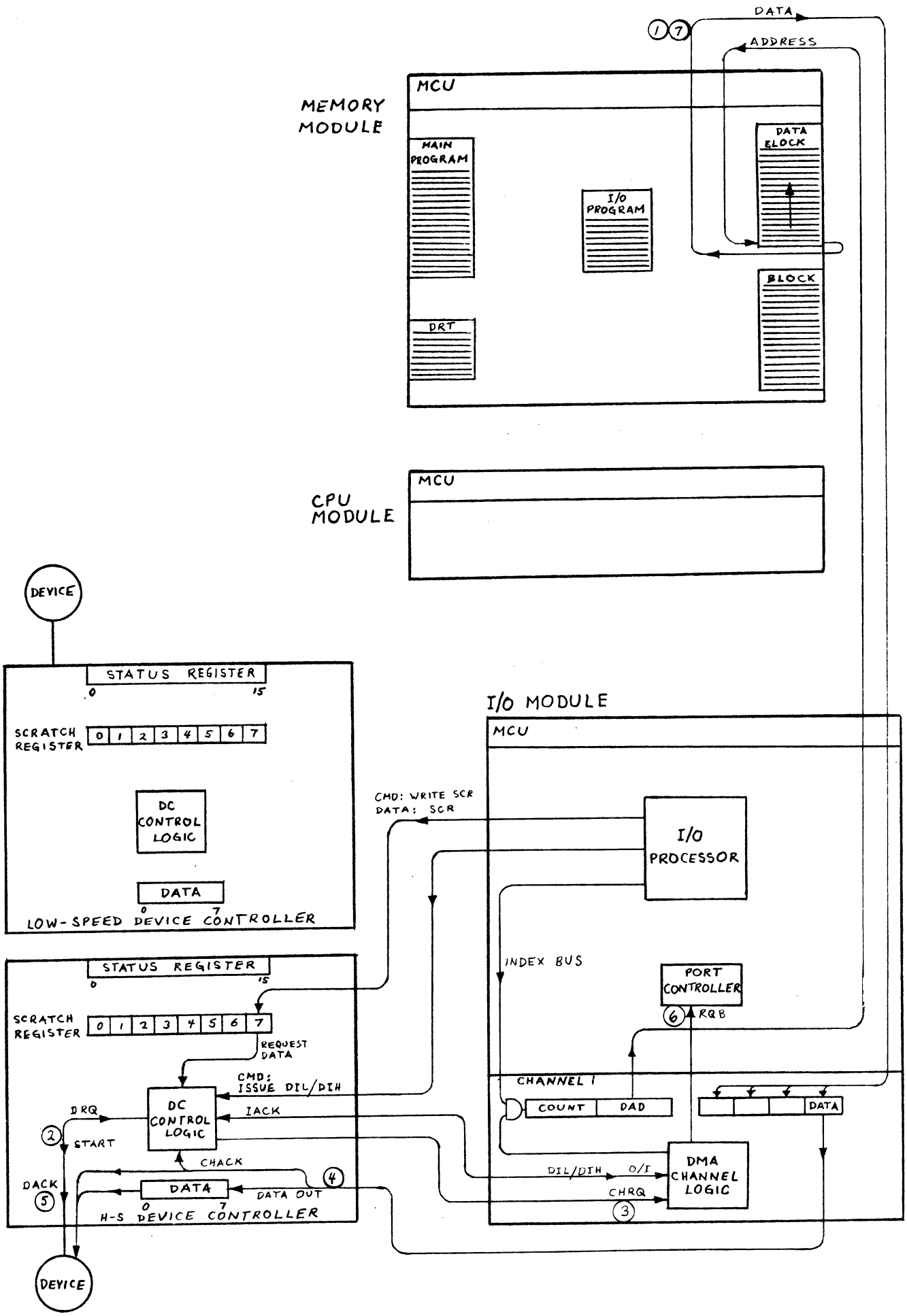
13. START action (preceding page, step 11) has caused a device-read, and DRQ (step 1 above) signifies that data is in the Data buffer.

14. After updating COUNT/DAD above, I/O Processor transfers data to itself. (CMD: Read Data.)

15. I/O Processor addresses memory with CBW (Clear Byte/Write).

16. I/O Processor sends data to memory.

17. Check if block done (10, 11, 12 above).


DATA
17
ADDRESS
MEMORY MODULE
MCU
MAIN PROGRAM
I/O PROGRAM
DATA BLOCK
BLOCK
DRT
CPU MODULE
MCU
DEVICE
STATUS REGISTER
0
15
SCRATCH REGISTER
0 1 2 3 4 5 6 7
DC CONTROL LOGIC
DATA
0
7
LOW-SPEED DEVICE CONTROLLER
I/O MODULE
MCU
CMD: WRITE SCR
DATA: SCR
I/O PROCESSOR
INDEX BUS
PORT CONTROLLER
6
RQB
STATUS REGISTER
0
15
SCRATCH REGISTER
0 1 2 3 4 5 6 7
REQUEST DATA
CMD: ISSUE DIL/DIH
DRQ
DC CONTROL LOGIC
IACK
2
START
CHACK
DACK
5
DATA
4
DATA OUT
0
7
H-S DEVICE CONTROLLER
CHANNEL 1
COUNT
DAD
DATA
DIL/DIH
O/I
DMA CHANNEL LOGIC
CHRQ
3
DEVICE

HIGH SPEED TRANSFERS

1. When the I/O Processor received its IODW (page 18), it initialized the Controller, which in turn initialized the Channel to which it is connected (step 10). The Channel therefore has COUNT/ADDRESS and direction (memory bound or device bound). If the direction is device bound, the Channel proceeds immediately to get the first word from memory, without waiting for the device to ask for it. Furthermore, the Channel contains a backup Data Buffer, so it can go ahead and get a second word while sending out the first word (in bytes) to the Device. (Device bound transfer described below.)

2. When the I/O Processor told the Controller to request Data transfer (page 18, step 11), the Controller issued START to the Device. The Controller is now waiting for DRQ (Device Request). The I/O Processor is from now on not involved.

OUT (DEVICE BOUND)

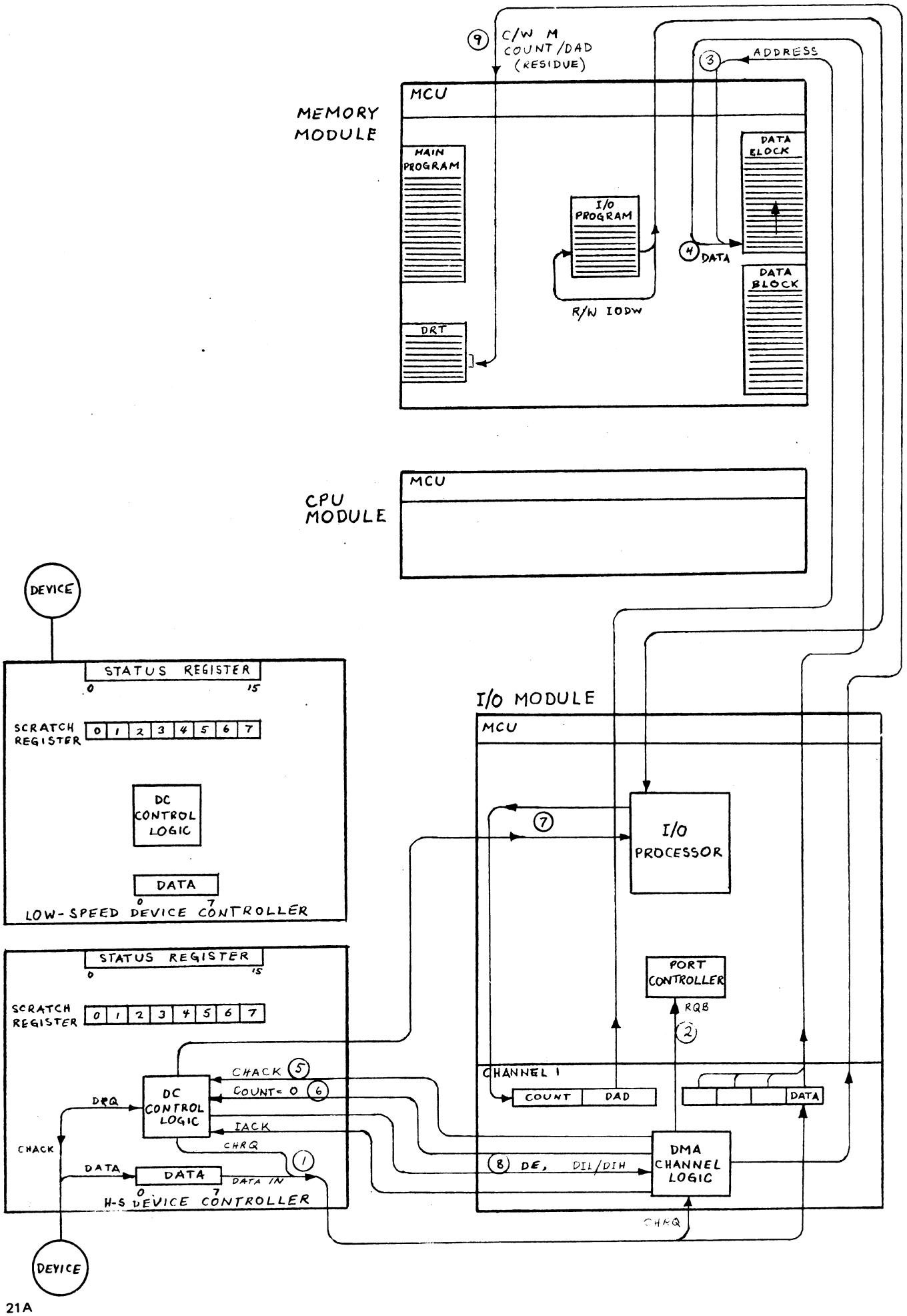
3. The Device begins the transfer by issuing DRQ to the Controller Logic, which relays the request to the Channel as CHRQ (Channel Request).

4. The Channel responds by sending out the first byte of data (low order byte first). The accompanying CHACK (Channel Acknowledge) tells the Controller "here is data".

5. CHACK is relayed to the Device as DACK (Device Acknowledge). This terminates the DRQ. After the Device has accepted the data and is ready for more, it re-issues DRQ. Steps 3, 4, and 5 are repeated until all 4 bytes have been transferred to the Device. Then proceed to step 6.

6. The fourth CHACK causes the Channel to increment DAD and decrement COUNT (if Inhibit Decrement is inactive) and to transfer its next data word (waiting in the Buffer) into the Shift Register. The Channel-to-Device transfer continues uninterrupted (steps 3, 4, 5). Meanwhile, the Channel initiates the procedure to refill the Buffer. The Channel logic continuously checks which memory module corresponds to Data Address, DAD, and whether that module is ready. If memory is ready, the Channel issues RQB (Request Bus) to the Port Controller for priority checks.

7. When selected, the Channel addresses memory (at the incremented DAD value), with a Read/Write memory opcode. When memory retrieves the data, it transmits the word to the Channel, where it is loaded into the Buffer.


MEMORY MODULE
MCU
MAIN PROGRAM
DRT
I/O PROGRAM
R/W IODW
DATA BLOCK
DATA BLOCK
9 C/W M COUNT/DAD (RESIDUE)
3 ADDRESS
4 DATA
CPU MODULE
MCU
DEVICE
STATUS REGISTER
0 15
SCRATCH REGISTER 0 1 2 3 4 5 6 7
DC CONTROL LOGIC
DATA
0 7
LOW-SPEED DEVICE CONTROLLER
I/O MODULE
MCU
7
I/O PROCESSOR
PORT CONTROLLER
RQB
2
CHANNEL 1
COUNT DAD
DATA
DMA CHANNEL LOGIC
8 DE, DII/DIH
CHRQ
STATUS REGISTER
0 15
SCRATCH REGISTER 0 1 2 3 4 5 6 7
CHACK 5
COUNT= 0 6
DRQ
DC CONTROL LOGIC
IACK
CHRQ
1
CHACK
DATA
DATA
DATA IN
0 7
H-S DEVICE CONTROLLER
DEVICE

HIGH SPEED TRANSFERS
(Continued)

IN (MEMORY BOUND)

1. On DRQ, Controller sends CHRQ and data to Channel. CHACK to Controller triggers read/DRQ cycle (4 times to load full word in Channel). Channel generates destination module number, and checks if that module is ready.

2. If destination module is ready, Channel issues RQB to Port Controller for priority checks. (Simultaneous LORQ and HIRQ.)

3. When selected, Channel's ADDRESS CYCLE sends C/W opcode to memory at DAD address. DATA CYCLE is set.

4. On second select from MCU, data is enabled for transmission to memory. DATA CYCLE terminates, increments DAD, and decrements COUNT.

5. CHACK to Controller ("data is in") causes another read cycle in the device. (Repeat back to step 1.)

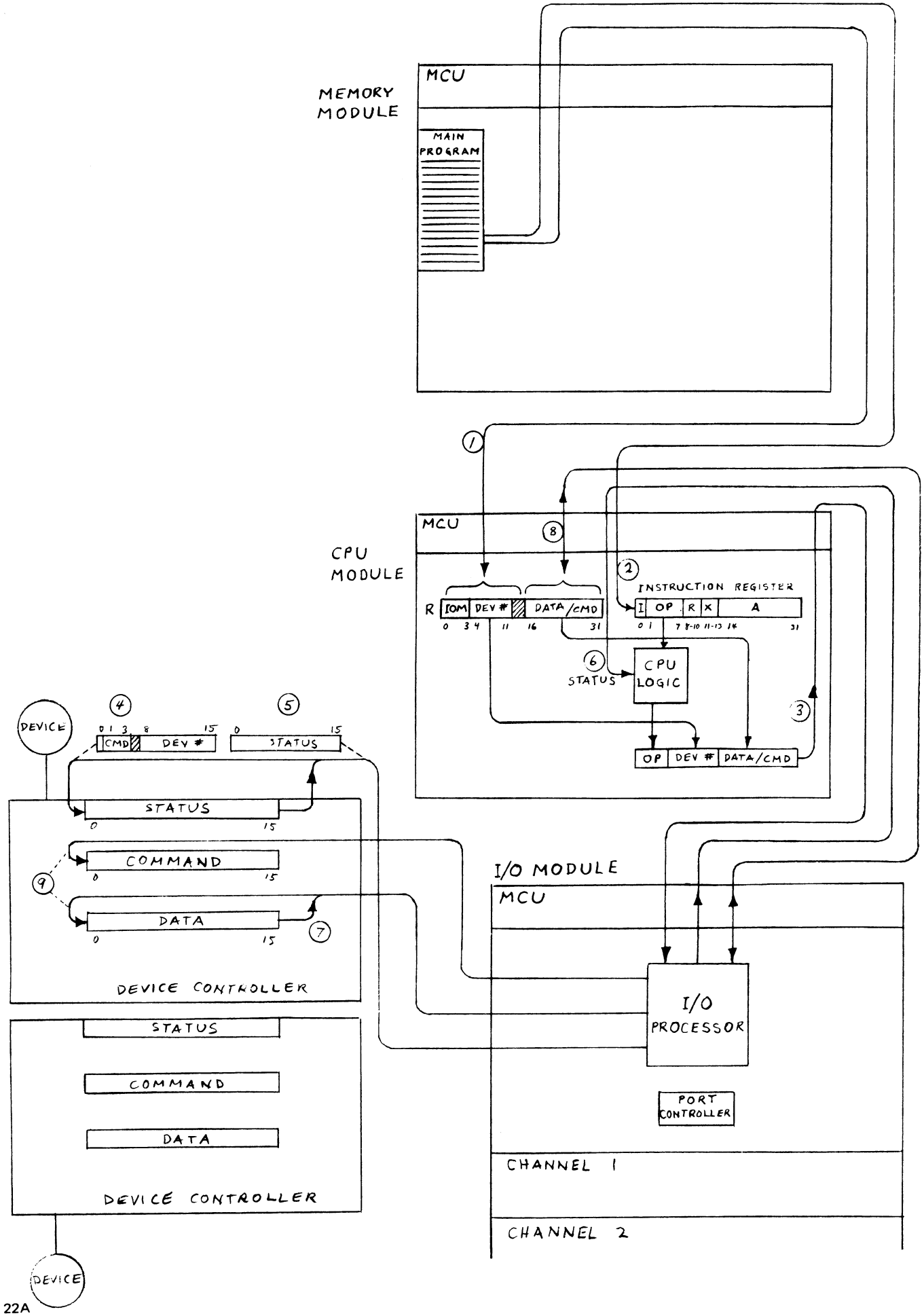
DATA CHAINING/END OF TRANSFER

6. When COUNT = 0, Channel loads its next IODW, if any, from its Backup Buffer (not shown) and tells the Controller to ask IOP for another IODW (SRLO signal).

7. If the I/O Processor is holding a Data Chaining bit from the previous IODW, it gets the new IODW and sends it to the Backup Buffer in the Channel. If the I/O Processor does not have a Data Chaining bit, proceed to step 8.

8. When Device signals End-of-Transfer, the Controller issues Device End, and the I/O Processor tells the Controller to issue DIL/DIH.

9. The result reads COUNT/DAD residue out to the second word of DRT. This tells the system how many words were transferred to/from where in memory. An Interrupt IOCW may be written into the I/O Program to notify CPU of transfer completion. Index Acknowledge, IACK, prevents the Controller from making further Channel Requests.


MEMORY MODULE
MCU
MAIN PROGRAM
CPU MODULE
MCU
INSTRUCTION REGISTER
R
IOM
DEV #
DATA/CMD
I
OP
R
X
A
CPU LOGIC
STATUS
OP
DEV #
DATA/CMD
DEVICE
CMD
DEV #
STATUS
STATUS
COMMAND
DATA
DEVICE CONTROLLER
STATUS
COMMAND
DATA
DEVICE CONTROLLER
DEVICE
I/O MODULE
MCU
I/O PROCESSOR
PORT CONTROLLER
CHANNEL 1
CHANNEL 2

OTHER I/O INSTRUCTIONS

The preceding pages illustrated the operation of the Start I/O (SIO) instruction. As mentioned earlier, there are four other I/O instructions:

RIO	Read I/O
WIO	Write I/O
CIO	Command I/O
TIO	Test I/O

The first three of these instructions transfer a 16-bit halfword between a CPU and a Device Controller. The TIO instruction simply transfers the Status halfword from the Device Controller to a CPU. Status is not returned for the CIO instruction (the only exception).

Following is the sequence of operations for the above instructions. Differences are noted within the text.

START INSTRUCTION SEQUENCE (ALL)

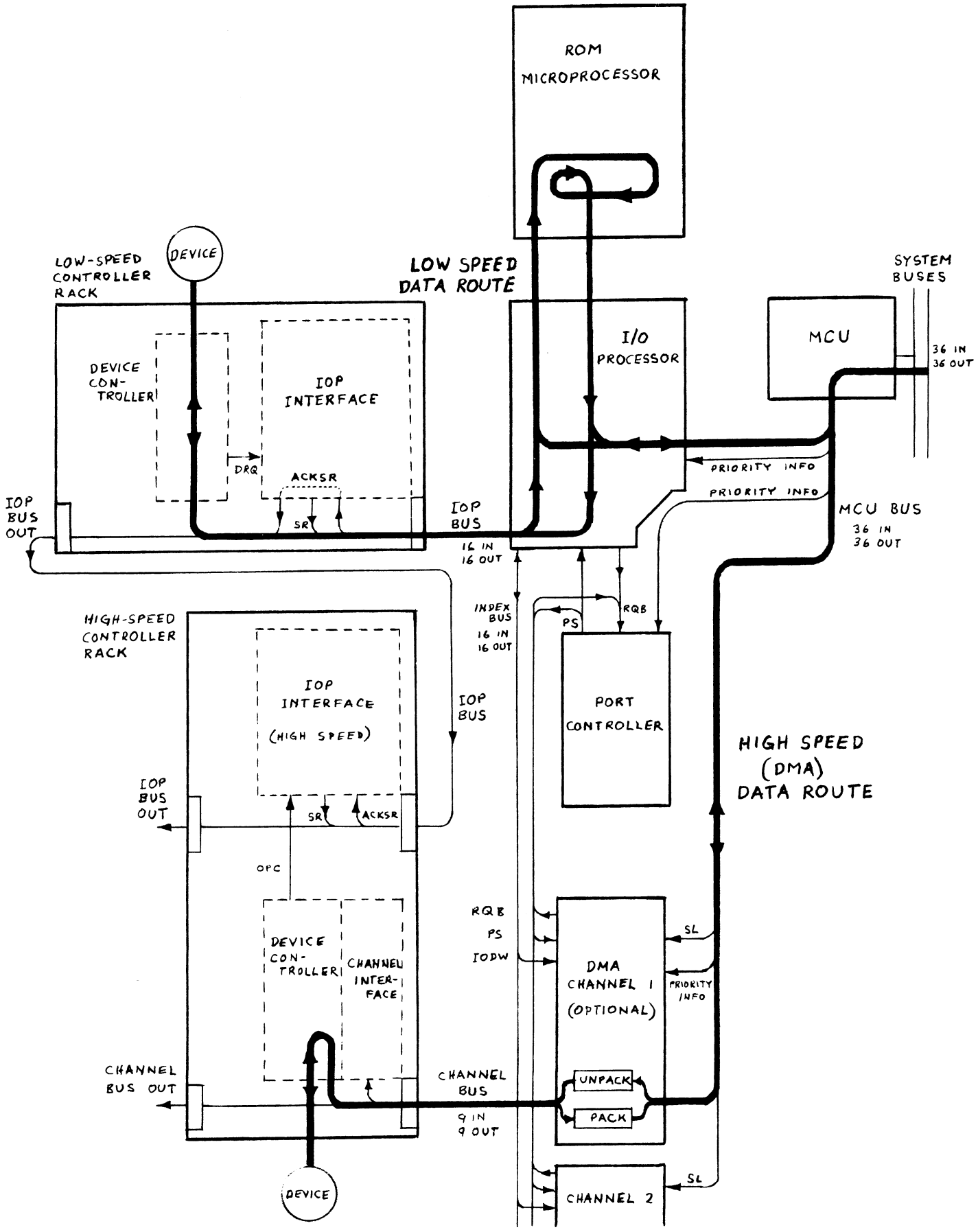
1. Program loads left half of R to identify the desired device and (if CIO or WIO) appropriate information in the right half.
2. Program issues instructions to a CPU.
3. CPU relays the instruction to the I/O Module.

CHECK STATUS (ALL EXCEPT CIO)

4. I/O Processor sends a "Read Status" command to the Controller.
5. Controller sends 16 bits of Status, with CMD ACK, to I/O Processor via Data-In lines. The I/O Processor begins its ROM program for the particular instruction.
6. If the instruction is TIO or WIO, Status is returned to the CPU. (TIO is complete at this point.) For RIO or WIO Status is checked before proceeding to the following steps.

TRANSFER HALFWORD (ALL EXCEPT TIO)

7. For RIO, I/O Processor sends a "Read Data" command to the Controller. Controller returns its Data register contents (up to 16 bits on the Data-In lines), with CMD ACK.
8. For RIO, data and Status are transmitted to the CPU, which loads the data into the right half of the R register specified in the R field of the RIO instruction. (RIO is complete at this point.)
9. For CIO or WIO, the I/O Processor places the data (up to 16 bits) on the Data-Out lines, and sends a "Write Command" (for CIO) or "Write Data" (for WIO) to the Controller. This loads the data into the Command or Data Registers, respectively.


ROM
MICROPROCESSOR
LOW SPEED
DATA ROUTE
SYSTEM
BUSES
LOW-SPEED
CONTROLLER
RACK
DEVICE
DEVICE
CON-
TROLLER
IOP
INTERFACE
DRQ
ACKSR
SR
I/O
PROCESSOR
MCU
36 IN
36 OUT
PRIORITY INFO
PRIORITY INFO
MCU BUS
36 IN
36 OUT
IOP
BUS
OUT
IOP
BUS
16 IN
16 OUT
INDEX
BUS
16 IN
16 OUT
RQB
PS
HIGH-SPEED
CONTROLLER
RACK
IOP
INTERFACE
(HIGH SPEED)
IOP
BUS
PORT
CONTROLLER
HIGH SPEED
(DMA)
DATA ROUTE
IOP
BUS
OUT
SR
ACKSR
OPC
RQB
PS
IODW
DEVICE
CON-
TROLLER
CHANNEL
INTER-
FACE
DMA
CHANNEL 1
(OPTIONAL)
SL
PRIORITY
INFO
CHANNEL
BUS OUT
CHANNEL
BUS
9 IN
9 OUT
UNPACK
PACK
DEVICE
CHANNEL 2
SL

SUB-UNITS OF THE I/O MODULE

The basic I/O Module includes the following sub-units:

ROM MICROPROCESSOR (3 cards). This is a small, general purpose arithmetic logic unit, providing 16-bit processing with 16 basic instructions and 7 accumulators.

I/O PROCESSOR (3 cards). The I/O Processor logic cards adapt the ROM Microprocessor for the specific purpose of handling the Omega I/O. These logic cards plus the ROM Microprocessor are together frequently considered to comprise "the I/O Processor", for simplicity.

PORT CONTROLLER (1 card). Determines internal priority among the 7 available DMA Channels and the I/O Processor (the 8th port) for transmission on the system bus. (An Address Mapper board is necessary for each two ports used.)

MODULE CONTROL UNIT (2 cards). Provides information on the status of other system modules, so that the Port Controller and I/O Processor can determine their transmission priorities.

In addition to these 4 basic sub-units, provision is made for up to 7 Direct Memory Access (DMA) Channels (2 cards each). These channels provide a data route that bypasses the I/O Processor. Each Channel keeps track of transfer counting and memory addressing, packs and unpacks full words for bus transmission, and is capable of concurrently handling the device transfers for one word while doing the memory transfer of another word. All of these factors contribute to an achievable transfer rate of 5 million 32-bit words per second (dependent on device capability and system configuration), with a worst case rate of about 500,000 wps (compared with 100,000 bytes per second at best through the I/O Processor).

The block diagram on the facing page shows the relationships of the 5 sub-units of the I/O Module (right side) and 2 external Device Controller racks. Heavy lines illustrate data routes for high-speed and low-speed transfers. Lighter lines show some of the major control signals.

Each of the sub-units will be discussed in detail in the following 10 pages. Briefly, however, the overall procedures for transferring data between peripheral devices and the System Buses are as follows. (Assume that the Devices are input devices.)

LOW SPEED TRANSFERS

When the Device has read its data, it transfers that data into a buffer in the Device Controller. A Data Request signal (DRQ) tells the IOP Interface to proceed with a Service Request (SR) to the I/O Processor.

The I/O Processor inputs this Service Request to the ROM Microprocessor (which may be operating some unrelated program segment). When ready to process the SR, the Microprocessor tells the I/O Processor to send out an ACKSR (Acknowledge Service Request). This signal propagates in series through the IOP Interface of each Device Controller. The first Controller encountered that has a set SR gets highest priority, and stops the ACKSR propagation.

Since the I/O Processor does not yet know what kind of service is requested, nor which Device is requesting it, the first message to be transmitted on the IOP Bus is a 16-bit word to the I/O Processor: 8 bits of request identification ("Scratch Register" contents) and 8 bits of Device Address.

With this information, the I/O Processor looks up the appropriate program in ROM and addresses a "Read Data" command (in this example) to the Controller. The Controller now sends the data byte to the I/O Processor, through the Microprocessor.

A Request-Bus signal (RQB) to the Port Controller is checked for priority with the other 7 ports (DMA Channels). Priority is assigned by plug adapters, and the I/O Processor may be assigned any one of the 8 positions.

When the Port Controller determines that no DMA transfer is in progress and that the I/O Processor is the highest priority port, a Priority-Select signal (PS) is sent to the I/O Processor. This enables the Module priority checking logic, using priority information of other system modules, supplied by the MCU. (The I/O Processor assumes some of the functions that normally occur within an MCU.)

Then, when the I/O processor determines that the I/O Module is the highest priority module requesting to transmit, the data is sent in on the System Buses.

HIGH SPEED TRANSFERS

High speed (DMA) transfers begin in the same way as low speed transfers. That is, the Controller must tell the I/O Processor what kind of service it wants. Repeating this part of the procedure in brief:

a. Data is received in the Device Controller.

b. Controller says "Operation Complete" (OPC).

c. Service Request (SR) is sent to I/O Processor.

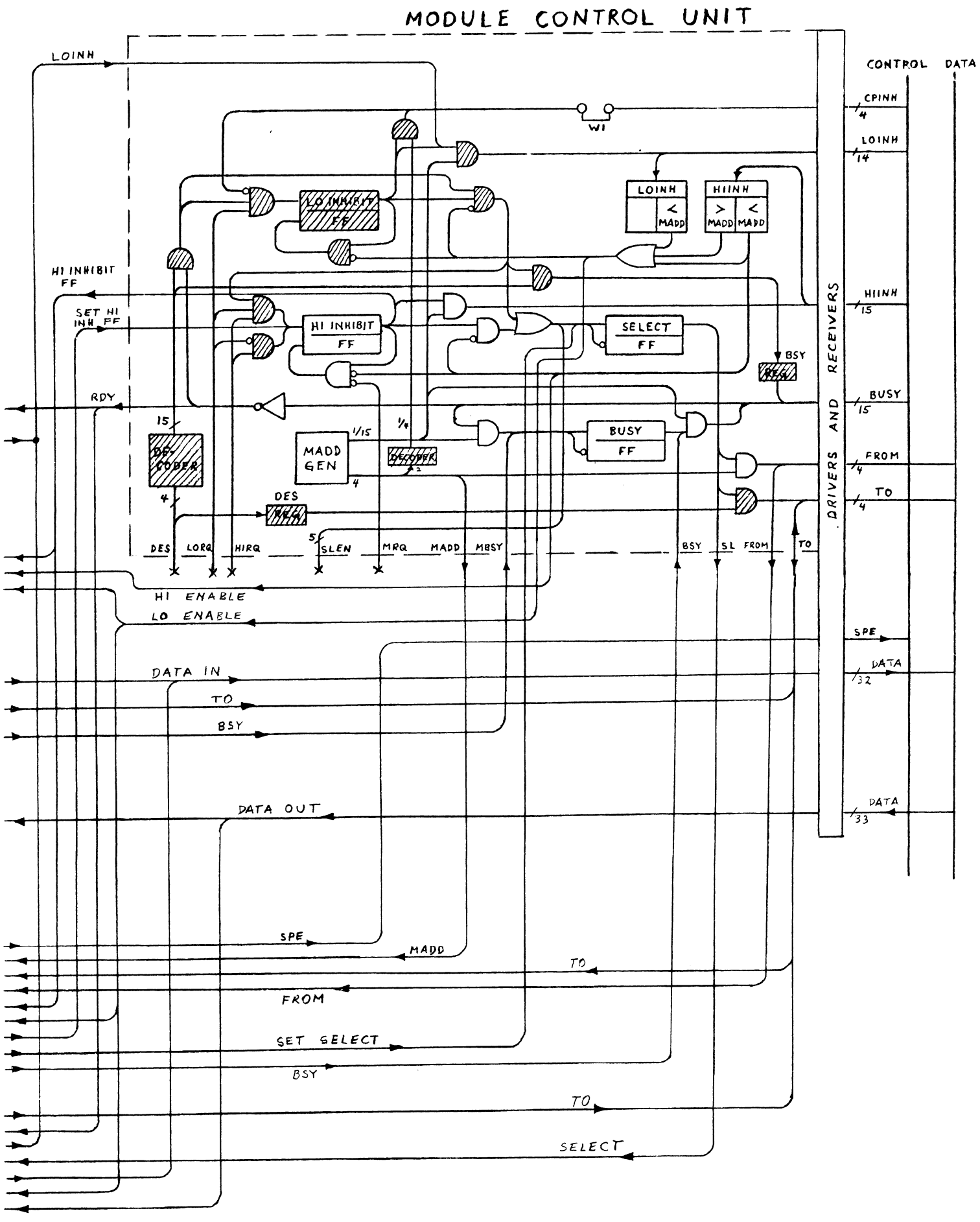
d. ACKSR is returned when I/O Processor is ready.

e. ACKSR in Controller reads out its Scratch Register.

Part of the Scratch Register information says that a high speed transfer is requested. This prompts the I/O Processor to get an IODW from memory and send it via the Index Bus to the appropriate channel. (Since the IODW is 32 bits, two transmissions on the Index Bus are necessary.)

On receipt of the IODW, the Channel operates directly with the Device Controller (bypassing the I/O Processor), and packs 4 bytes into a buffer in the Channel. The Channel then asks the Port Controller for priority (RQB), which responds when ready with Priority Select (PS).

When the Channel determines that it may transmit (using priority information from MCU), it makes a low-priority Address transmission to memory, followed by a high-priority Data transmission. The Channel maintains a count of how many words it has transferred, and increments the Address as necessary.


MODULE CONTROL UNIT
LOINH
WI
LO INHIBIT FF
LOINH < MADD
HIINH > MADD < MADD
HI INHIBIT FF
SET HI INH FF
HI INHIBIT FF
SELECT FF
BSY
RDY
15
DE-CODER
4
MADD GEN
1/15
1/4
DECODER
2
4
BUSY FF
DES
REG
5
DES LORQ HIRQ SLEN MRQ MADD MBSY
BSY SL FROM TO
HI ENABLE
LO ENABLE
DATA IN
TO
BSY
DATA OUT
SPE
MADD
TO
FROM
SET SELECT
BSY
TO
SELECT
DRIVERS AND RECEIVERS
CONTROL
DATA
CPINH 4
LOINH 14
HIINH 15
BUSY 15
FROM 4
TO 4
SPE
DATA 32
DATA 33

MODIFIED MCU

When used in the I/O Module, the Module Control Unit is modified to furnish 3 additional outputs and 2 additional inputs. These are:

Outputs: Hi Inhibit FF

Hi Enable ("no higher HIINH present in system")

Lo Enable ("no higher LOINH, no HIINH present")

Inputs: Set Hi Inhibit FF

Set Select FF

These modifications allow various parts of the I/O Module to do their own determinations of module priority.

Those logic elements shown shaded in the facing diagram are never used, in this application of the MCU.

Otherwise, the detailed operation of the MCU is the same as discussed earlier in this manual.

NOTE

This diagram and the 8 following logic diagrams are specially arranged so they may be joined together in the exact layout illustrated by the overall block diagram on the preceding page.

Coding Table — ROM Microprocessor

FIELD:	FUNCTION SELECT				R BUS SELECT				S BUS SELECT		ROTATE SELECT			DESTINATION SELECT		
Bit:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binary ↓																
0	NOP				ZEROS				ZEROS		NO SHIFT			NO STORE		
1	NOP				AC1				AC5		L1			AC1		
2	BRT				AC2				AC6		R1			AC2		
3	BRF				AC3				EXT		XA			AC3		
4	IOR				AC4						XB			AC4		
5	XFS				CA1						OT			AC5		
6	XFR				ACQ						XC			AC6		
7	AND				CA1						DL			ACQ		
8	ADD				AQ0											
9	–				AQ1											
10	INC				AQ2											
11	–				AQ3											
12	XOR				AQ4											
13	CMS				AQ5											
14	CMR				AQ6											
15	CLT				AQ7											

OPTIONAL FIELD:		CONDITION SELECT	BRANCH ADDRESS SELECT		OUTPUT SELECT
0		ZERO	ZERO		OT1
1		CF1	IMMED		OT2
2		CF2	AC3		OT3
3		–	IMMED OR AC3		OT4
4		CF3			OT5
5		–			OT6
6		–			OT7
7		–			OT8
8		ZERO			
9		CF4			
10		CF5			
11		CF4			
12		CF6			
13		–			
14		–			
15		–			

ROM MICROPROCESSOR

The external functions performed by the ROM Microprocessor are dependent on the application, or more specifically, on the micro-programs written and permanently stored in the Read-Only Memory (ROM). In this context, the application is assumed to be the control of Omega I/O. As part of the "I/O Processor", the Microprocessor looks for an SIO instruction from a CPU, relative to a specific device, then branches to a routine that begins the I/O service. Some of the initial operations performed are to extract the device number from the CPU instruction, get Status from that device and return it to the CPU, and (if the device's status indicates that it is ready to start I/O) tell the device to request its first program word (IOCW). These operations (the exact sequences were discussed earlier), are external functions of the micro-programs in ROM.

However, the internal functions of the ROM Microprocessor are general in nature. The Microprocessor is actually a small computer, containing the usual Instruction Register, Arithmetic Logic Unit, Accumulators (7), and R-S-T bus system. It has an instruction set of 13 basic instructions, plus 7 micro-instructions for rotate-shift options. Instruction coding is shown on the facing page; definitions are as follows.

FUNCTIONS (0-3)

NOP	No operation
BRT	Branch on selected condition True, to selected Branch Address
BRF	Branch on selected condition False, to selected Branch Address
IOR	Inclusive OR of R and S buses onto T bus
XFS	Transfer S bus onto T bus
XFR	Transfer R bus onto T bus
AND	Logical AND of R and S buses onto T bus
ADD	Arithmetic addition of R and S buses onto T bus
INC	Arithmetic addition and increment (R + S + 1) onto T bus
XOR	Exclusive OR of R and S buses onto T bus
CMS	Complement of S bus onto T bus
CMR	Complement of R bus onto T bus
CLT	Clear T (zeros onto T bus)

R BUS/CONDITION (4-7)

ZEROS	Zeros forced onto R bus
AC( )	AC( ) enabled onto R bus
CA1	Conditional AC1. If external (wired) condition is true, AC1 enabled onto R bus; if condition is false, zeros forced onto R bus.
AQ( )	ACQ enabled onto R bus, and one of 8 (numbered) inputs replaces old contents of ACQ. Number 0 (AQ0) puts old contents of ACQ back into ACQ. If bits 13-14-15 select ACQ as a destination, replacement input selected by AQ( ) is overridden.
ZERO	Forces False condition, for unconditional BRF branching
CF( )	Selects numbered Condition Flip-flop for conditional branching

S BUS/BRANCH ADDRESS (8-9)

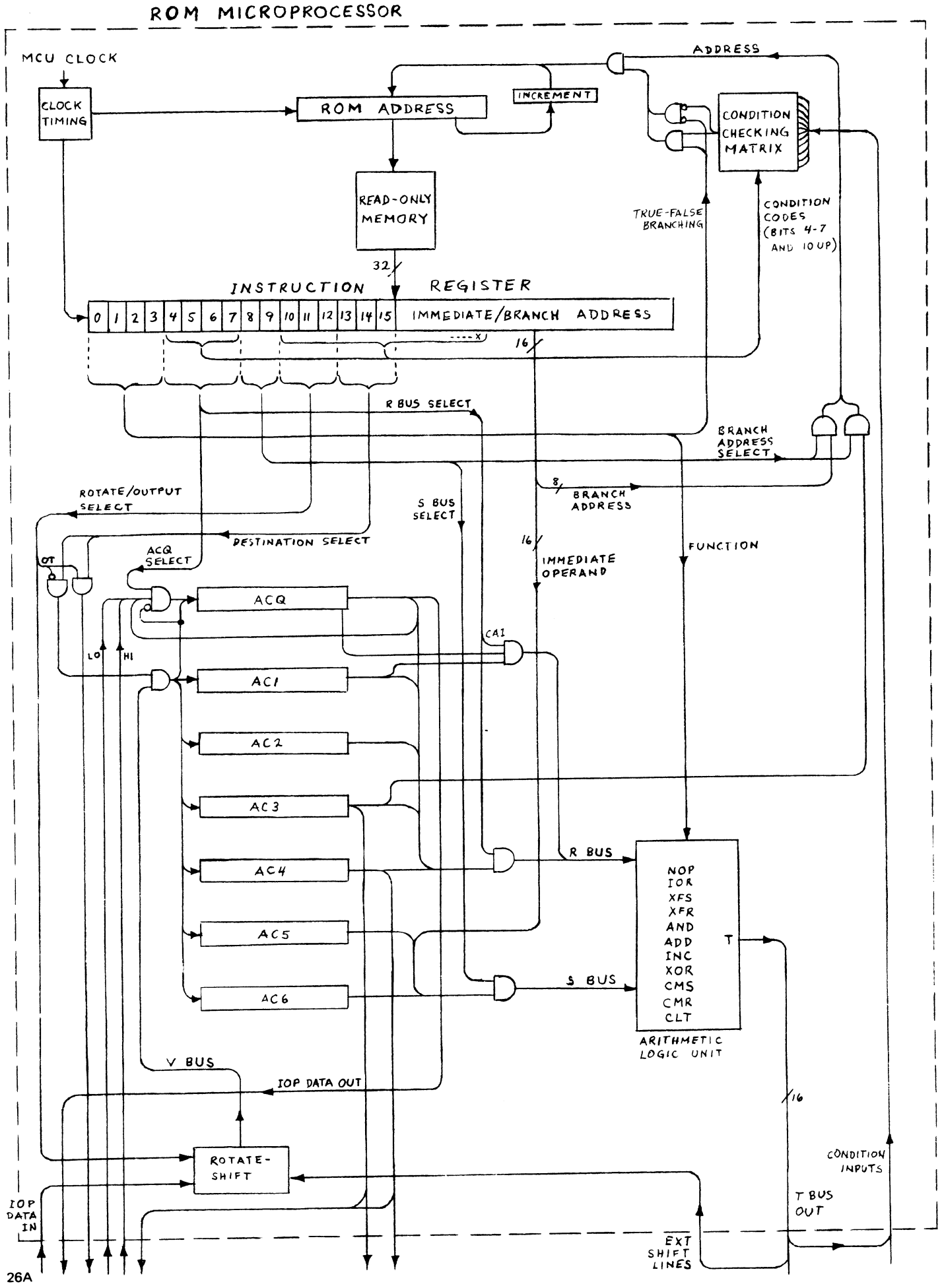
ZEROS	Zeros forced onto S bus
AC5/6	AC5 or AC6 enabled onto S bus
EXT	Immediate operand onto S bus
ZERO	Branch to address Zero
IMMED	Branch to address specified in Operand field
AC3	Branch to address specified in AC3 register
IMMED OR AC3	Branch to address resulting from inclusive OR of Operand field and AC3 contents

ROTATE (10-12)

ZEROS	No shift
L1	Rotate left one onto V bus
R1	Rotate right one onto V bus
XA/B	Select rotate option A or B
OT	Send 3-bit Output code (bits 13-14-15) to external logic for selectable destination of T bus
XC/DL	If external (wired) condition is true, a left-one shift is made (Divide Left); if condition is false, rotate option C is selected. (Machine does not distinguish between the binary codes for XC and DL; selection is made only by the input condition.)

DESTINATION/OUTPUT (13-15)

ZEROS	No store
AC( )	Store V Bus into AC( ) register
OT( )	Store T Bus into one of 8 external destinations; enabled by OT code of bits 10-11-12.


ROM MICROPROCESSOR
MCU CLOCK
CLOCK TIMING
ROM ADDRESS
INCREMENT
ADDRESS
CONDITION CHECKING MATRIX
READ-ONLY MEMORY
TRUE-FALSE BRANCHING
CONDITION CODES (BITS 4-7 AND 10 UP)
32
INSTRUCTION REGISTER
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
IMMEDIATE/BRANCH ADDRESS
16
R BUS SELECT
BRANCH ADDRESS SELECT
8
BRANCH ADDRESS
ROTATE/OUTPUT SELECT
S BUS SELECT
DESTINATION SELECT
ACQ SELECT
OT
16
IMMEDIATE OPERAND
FUNCTION
ACQ
CAI
LO
HI
AC1
AC2
AC3
AC4
AC5
AC6
R BUS
S BUS
NOP
IOR
XFS
XFR
AND
ADD
INC
XOR
CMS
CMR
CLT
T
ARITHMETIC LOGIC UNIT
V BUS
IOP DATA OUT
16
ROTATE-SHIFT
CONDITION INPUTS
T BUS OUT
IOP DATA IN
EXT SHIFT LINES
26A

MICROPROCESSOR LOGIC

READ-ONLY MEMORY

The capacity of the ROM is expandable, but in the basic I/O Processor configuration it provides 256 words of 32 bits each. Addressing therefore requires 8 of the available 16 address bits of the instruction word. The ROM address is incremented automatically after each instruction, unless overridden by a new address from AC3 register or from the operand field. (The "or" of both sources is also possible, as well as address zero; see previous page.)

Conditional branching is determined by a Condition Checking Matrix. Depending on Matrix wiring, up to 24 external Condition Inputs may be connected to the input of the Matrix. In the I/O Processor application, these inputs include Service Requests and Command Acknowledges from Device Controllers, and transmit and receive notifications. It is also possible to connect T bus bits, overflow, and carry, etc.; in some of these cases, however, there are multiplexing restrictions. (There are only 6 Condition Flip-flops in the Matrix, selectable by bits 4-7; each flip-flop can have up to four multiplexed input gates, selected by otherwise unused bits of the instruction word. Typically, bits 16-23 of the address field are used, but actually bits 10-23 are available for specifying Conditions, if not otherwise modified.)

INSTRUCTION REGISTER

Instruction words coming out of ROM are clocked into the Instruction Register at the rate of 5 MHz. The higher order 16 bits are the instruction code, and the lower order 16 bits are either an immediate operand or a Branch address, depending on the instruction. The diagram on the facing page shows the grouping of instruction bits and the logic elements they affect.

Bits 0-3 set up one of 11 functions in the arithmetic logic unit, or select whether a branch address is "go" if the tested condition is true or false (BRT or BRF).

Bits 4-7 can be used in one of 3 ways. If the function selected by bits 0-3 is BRT or BRF, bits 4-7 read out the state of one of the 6 Condition flip-flops. Otherwise, bits 4-7 determine either what goes onto the R bus (ACQ, AC1, AC2, AC3, AC4, or all zeros) or what goes into the ACQ register (self or external input) as a replacement value while simultaneously reading out the current ACQ value to the R bus. Only two external inputs are shown; these are the low order and high order halves of the 32-bit words received from other modules via the system Data bus. The wiring for conditionally reading out AC1 to the R bus (CA1) shows bit 15 of ACQ as the required condition; this configuration is for multiplication. However, for other applications the condition input could come from any other register or external source.

Bits 8 and 9 can be used in one of 2 ways. If bits 0-3 selected BRT or BRF, bits 8 and 9 determine what gets used as a new ROM address: the low order bits from the current instruction word (8 bits or more, depending on ROM capacity), or the contents of AC3, or the logical "or" of both sources, or a forced address of zero. If bits 0-3 do not specify BRT or BRF, bits 8 and 9 determine what goes onto the S bus: AC5, AC6, all zeros, or "External". In this case, External is wired to be the Immediate Operand from the Instruction Register.

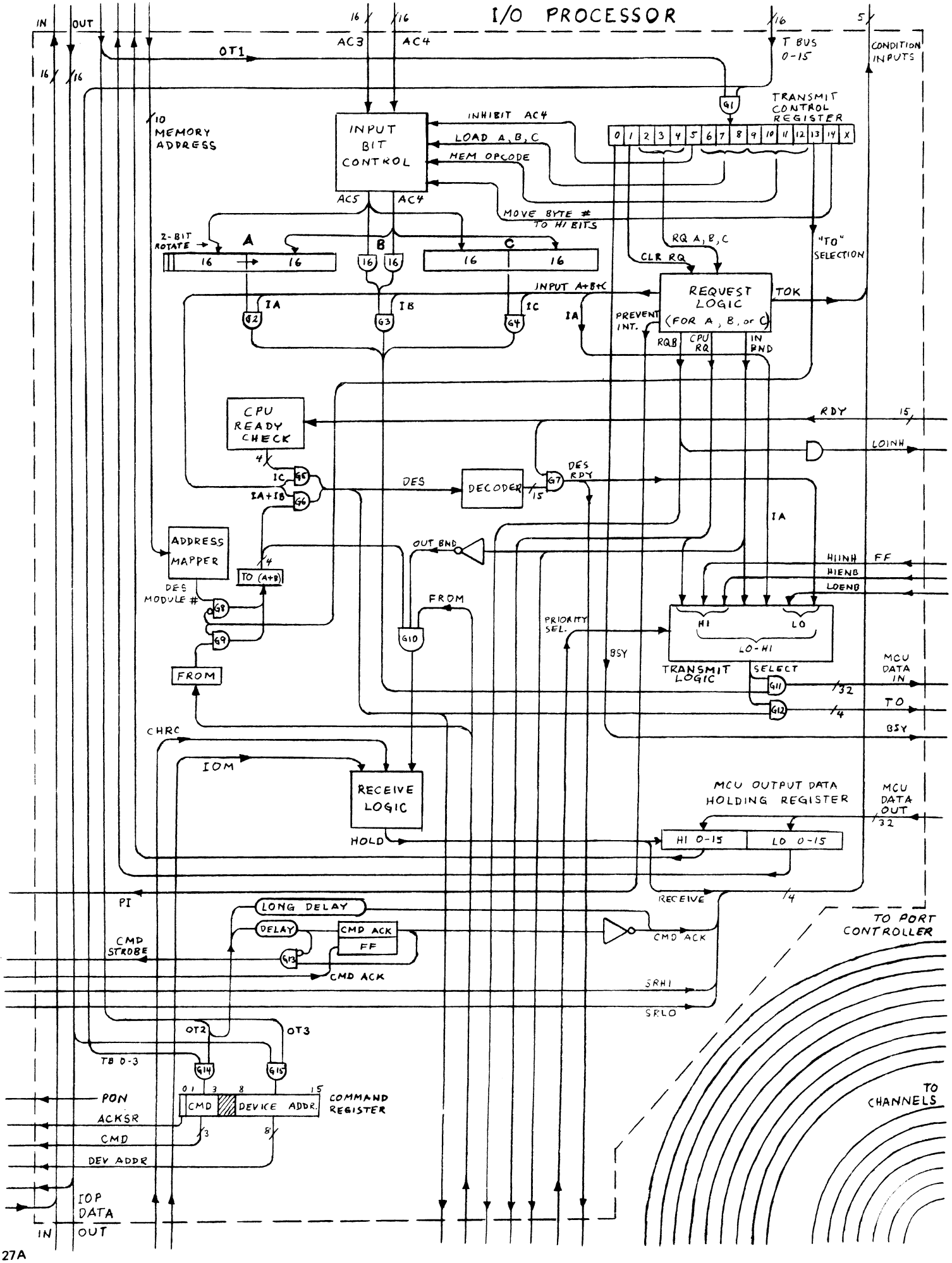
Bits 10-12, except when coded "OT", enable one of seven hard-wired rotate/shift options. Inputs from either an internal source (T bus) or an external source (IOP Data In, in this case) are rotated or shifted out onto the V bus. Definitions of the options were given on the preceding page. For the I/O Processor application, assignments of the options are as follows: No-Shift, L1, and R1 use the T bus input. Option A is a left-four rotate of the T bus, Option B is a left-two rotate of the T bus, with 2-bit extend, and Option C routes IOP Data In unshifted to the V bus. DL is not used. When bits 10-12 are coded "OT", Destination bits 13-15 are gated out of the Microprocessor (for external routing control of output data) instead of, as normally, to select one of the accumulators for a V bus destination.

ACCUMULATORS

Of the seven 16-bit accumulators, ACQ is the only one that outputs and inputs directly from external sources. It may be selected as a V bus destination by bits 13-15, or (not in the same instruction) it may be read out onto the R bus by bits 4-7, with or without the "replacement" feature discussed earlier. AC1 through AC4 are R bus sources, and AC5 and AC6 are S bus sources. Additionally, AC1 has the "conditional-read" feature, and AC3 is a ROM address source. Together AC3 and AC4 provide 32-bit output for transmission of full words to other modules. If the transmission type happens to be an address to memory, bits 12-21 (most significant 10 address bits) are also sent to an Address Mapper in the I/O Processor.

ARITHMETIC LOGIC UNIT

The ALU is conventional, executing one of 11 functions on R and/or S bus inputs, routed out on the T bus. Externally, the T bus may be wired as desired. For the I/O Processor application, the T bus is connected directly back in to the Rotate-Shift unit, and an "or" of all 16 bits is used as a condition input. Other applications might "pre-shift" T bus bits before application to the Rotate-Shift unit, and specific bits, such as sign or least significant bit, might be wired as condition inputs.


I/O PROCESSOR
IN
OUT
16
16
AC3
AC4
OT1
16
16
T BUS 0-15
5
CONDITION INPUTS
G1
TRANSMIT CONTROL REGISTER
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 X
10 MEMORY ADDRESS
INPUT BIT CONTROL
INHIBIT AC4
LOAD A, B, C
MEM OPCODE
AC5
AC4
MOVE BYTE # TO HI BITS
2-BIT ROTATE
A
B
C
16
16
16
16
16
16
RQ A, B, C
CLR RQ
"TO" SELECTION
INPUT A+B+C
REQUEST LOGIC (FOR A, B, or C)
TOK
PREVENT INT.
RQB
CPU RQ
IN BND
IA
IB
IC
G2
G3
G4
IA
CPU READY CHECK
4
IC
IA+IB
G5
G6
RDY
15
LOINH
DES
DECODER
15
DES RDY
G7
ADDRESS MAPPER
DES MODULE #
G8
G9
4
TO (A+B)
FROM
OUT BND
G10
FROM
IA
HIINH
HIENB
LOENB
FF
PRIORITY SEL.
HI
LO
LO-HI
BSY
TRANSMIT LOGIC
SELECT
G11
G12
32
4
MCU DATA IN
TO
BSY
CHRC
IOM
RECEIVE LOGIC
HOLD
MCU OUTPUT DATA HOLDING REGISTER
HI 0-15
LO 0-15
MCU DATA OUT
32
RECEIVE
4
PI
LONG DELAY
DELAY
CMD ACK
FF
G13
CMD ACK
CMD STROBE
CMD ACK
SRHI
SRLO
TO PORT CONTROLLER
OT2
OT3
TB 0-3
G14
G15
0 1 3 8 15
CMD
DEVICE ADDR.
COMMAND REGISTER
PON
ACKSR
CMD
DEV ADDR
3
8
TO CHANNELS
IOP DATA
IN
OUT

I/O PROCESSOR LOGIC

The top half of the facing diagram is the transmit control logic. The lower half shows the receive logic and the device command logic.

TRANSMIT CONTROL LOGIC

Logic elements of the upper half of the diagram are responsible for all transmissions to other modules except high-speed data transfers.

There are 3 categories of transmission modes, arbitrarily labeled A, B, C. Mode A rotates the 32-bit output of accumulators AC5 and AC4 two bits to the right, before loading into the A input register. This is for memory addressing, and moves the byte number from right end to the left end, where it forms part of the memory opcode. (Memory opcodes are listed on page 11A.)

Mode B provides un-rotated input (such as for data) for most other transmissions. The exception is Mode C, which is strictly for interrupts to a CPU. Since the CPU might not accept an interrupt for appreciable lengths of time (a "ready" check is made at gate G5), the interrupt word is held in the C input register so that the I/O Processor may proceed with other operations while waiting for the CPU to become ready.

The transmission mode is determined by the Microprocessor, by strobing a 15-bit control word from the T Bus (OT1 strobe) into the Transmit Control Register. Various bits of this word set up or clear the A, B, or C transmission mode, set the module "busy", limit the transmission to 16 bits (Inhibit AC3), insert the memory opcode into a memory addressing word, and determine whether "TO" should be the same as the previous "FROM" (i.e., a return transmission) or be determined by the Address Mapper. (The Address Mapper has plug adapters that allow a memory module number to be decoded from the 10 most significant bits of an absolute memory address.)

The Transmit Logic is capable of providing Select for 3 types of MCU transmissions: HI-priority, LO-priority, and dual LO-HI transmissions. A LO-priority Select is made if the destination is ready (checked at G7) and no HIINH or higher-priority LOINH exists in the system (LOENB input from MCU). A HI-priority Select is made (for CPU Requests) if the Port Controller has set the Hi Inhibit FF in MCU, and if there is no higher-priority HIINH. Two Selects occur for In-Bound, non-CPU transfers, if both A and B Modes are enabled and the appropriate priority inputs are present (DESRDY and LOENB for LO-priority Select, and HIINH FF and HIENB for HI-priority Select).

The Select signal at G11 sends the data to the Data Bus, and at G12 it sends the TO code. The FROM code automatically is sent out for all I/O Module transmissions by a Select in the MCU.

RECEIVE LOGIC

Data intended for output to device is received in a 32-bit Holding Register. It is locked in the register by a Hold signal, if receive conditions are met.

Receive conditions are met if the data is not claimed by a DMA Channel (Channel-Receiving signal CHRC), if the TO code is for this module (IOM), and if this module is waiting for this data (FROM must compare with the TO code retained at G10 input).

A Receive signal (Hold) is applied as a Condition input to the Microprocessor, informing that the data has been received. Two 16-bit halves of the received data are available to the Microprocessor at separate ACQ inputs.

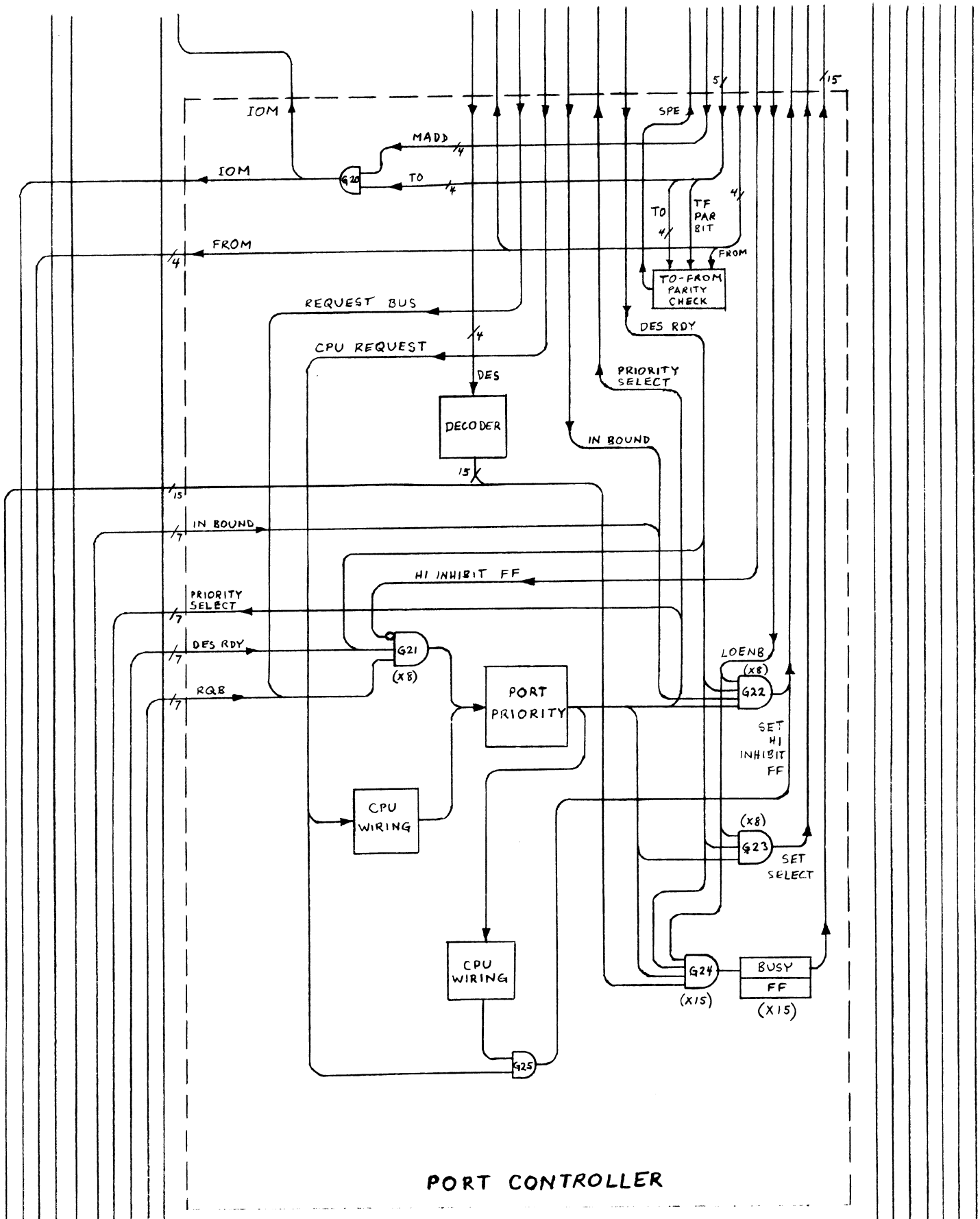
DEVICE COMMAND LOGIC

The Service Requests from the Device, SRHI and SRLO, are routed directly through the I/O Processor logic to the ROM Microprocessor (top). The Microprocessor responds with a 16-bit command word. An accompanying OT2 strobe loads 4 command bits into the Command Register via G14, and an OT3 strobe loads 8 bits of Device Address via G15. All bits go out to all Device Controllers.

A delay at the input of G13 allows time for the command to be accepted by the input logic of the Device Controller. Then, the Command Acknowledge flip-flop is set (now waiting for CMD ACK), and G13 sends a Command Strobe to all Controllers.

The Controller which has been addressed will respond to the Command Strobe by returning CMD ACK, meaning that the Controller has received its command and is now processing it. This clears the CMD ACK flip-flop, which relays the acknowledge to the Microprocessor.

If, for any reason, the Device Controller does not respond, the I/O Processor itself generates a pseudo CMD ACK ("Long Delay"). By the absence of other information (e.g., no data on the input lines), the ROM program can detect the failure and proceed accordingly.



PORT CONTROLLER BUS
TO CHANNELS

MCU BUS
TO CHANNELS

PORT CONTROLLER

The Port Controller performs the following functions:

SELECTS HIGHEST-PRIORITY RQB

Each time a DMA Channel or the I/O Processor wants to transmit information to another module, it applies an RQB (Request Bus) signal to the Port Controller. The Port Priority logic accepts these RQB inputs, and causes the RQB with the highest pre-assigned priority (from 1 to 8) to inhibit those with lower priority. The selected RQB becomes a Priority Select (PS) output, on one of the 8 PS lines.

SETS UP TRANSMISSION TYPE

The Port Controller uses two input signals to determine transmission type: In-Bound (true or false) and CPU Request (true or false). Using this information, one of 3 types of bus transmissions is initiated:

a. INBOUND. This requires a LO-priority transmission (memory address and opcode), followed by a HI-priority transmission (data). Provided that the destination is ready and the I/O Module has highest priority (DESRDY and LOENB input terms to G23), a Set Select signal is sent out to MCU. Setting the MCU's Select flip-flop transmits the FROM code, and the data and TO code of a priority-selected DMA Channel. (The I/O Processor uses Priority Select to generate its own Select for data and TO.) The In-Bound signal at G22 additionally sends a set signal to the MCU's High Inhibit flip-flop, causing the MCU to initiate a HI-priority transmission following completion of the current Select.

b. OUTBOUND. This requires only a LO-priority transmission (address and opcode to memory). This is the same as above except G22 is not enabled (In-Bound false), so that the HI Inhibit cycle does not get initiated.

c. INBOUND TO CPU. This requires only a HI-priority transmission (Status information to CPU). A CPU Request signal from the I/O Processor is applied directly (via "CPU Wiring") to Port Priority, instead of an RQB to G21. (HI-priority transmissions are not dependent on the DESRDY term.) The Priority Select output (via G25) directly sends a set signal to the MCU's Hi Inhibit flip-flop, bypassing G22. This avoids the LOENB and DESRDY terms.

SENDS "BUSY" TO DESTINATION

For each LO-priority transmission, G24 causes the appropriate Busy flip-flop (1 of 15) to issue a Busy signal to the destination module. As discussed earlier (under MCU, page 5), this makes the destination module go "busy" at the same time that it receives data.

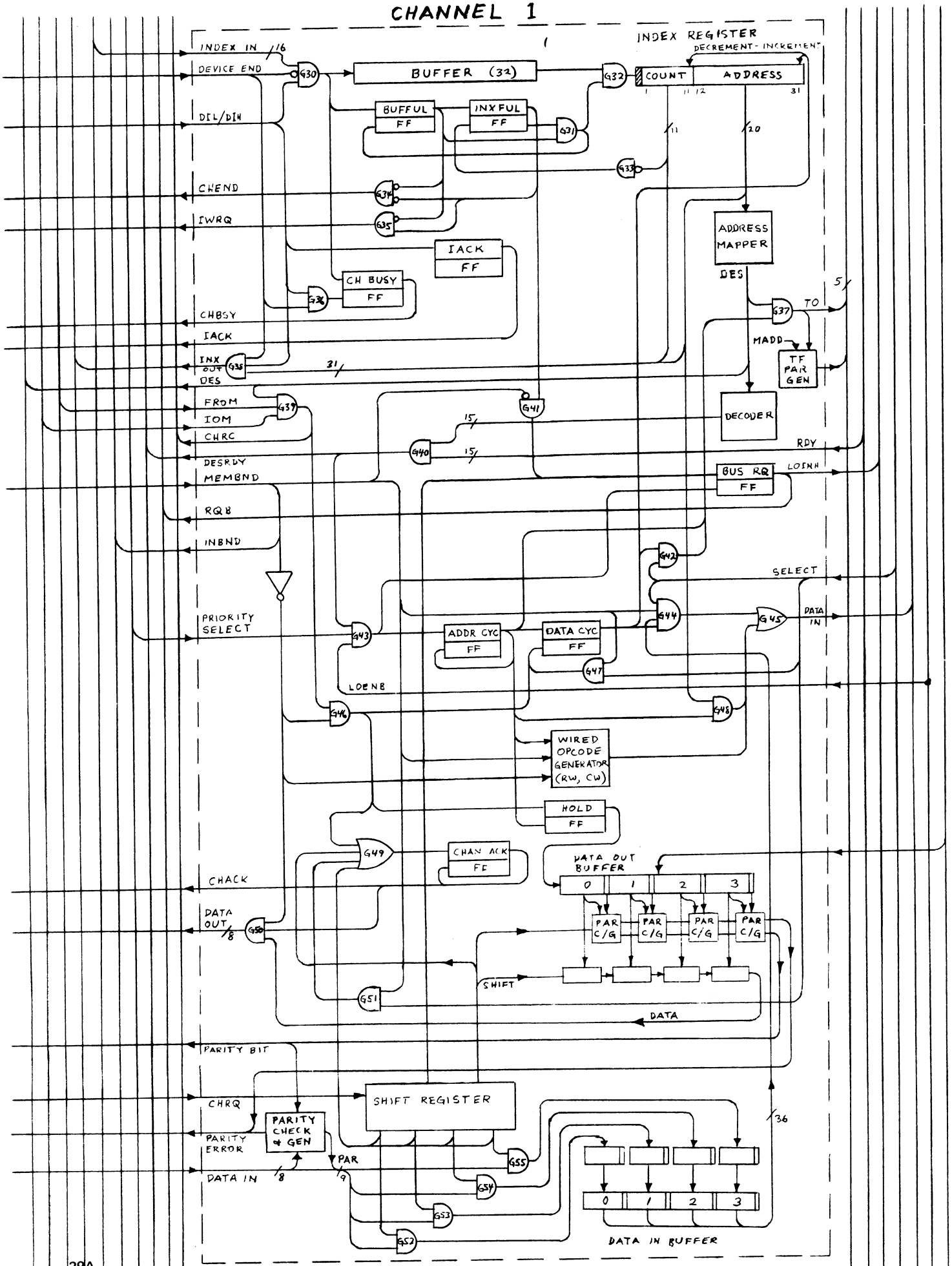
PROVIDES FROM AND IOM

The four FROM lines of the system's Data Bus are furnished to the I/O Processor and the DMA Channels, for the purpose of identifying an expected transmission from a specific module.

The IOM signal (I/O Module) is generated when the incoming TO code compares exactly with the address of "this module", as coded by plug adapters (MADD). IOM means that this module is currently being addressed.

CHECKS FOR SYSTEM PARITY ERROR

The incoming TO and FROM bits (total of 8 bits) are checked for proper parity with the accompanying To-From Parity Bit. If an error occurs (parity not "even"), a System Parity Error (SPE) is sent out on the system control bus. This signal indicates that a module-addressing error has occurred.


CHANNEL 1
INDEX REGISTER
DECREMENT - INCREMENT
COUNT
ADDRESS
INDEX IN
DEVICE END
DIL/DIN
CHEND
IWRQ
CHBSY
IACK
INX OUT
DES
FROM IOM
CHRC
DESRDY
MEMBND
RQB
INBND
PRIORITY SELECT
LOENB
CHACK
DATA OUT
PARITY BIT
CHRQ
PARITY ERROR
DATA IN
BUFFER (32)
BUFFUL FF
INXFUL FF
CH BUSY FF
IACK FF
HOLD FF
CHAN ACK FF
ADDR CYC FF
DATA CYC FF
BUS RQ FF
ADDRESS MAPPER
DECODER
TF PAR GEN
MADD
TO
RDY
LOINH
SELECT
DATA IN
WIRED OPCODE GENERATOR (RW, CW)
DATA OUT BUFFER
PAR C/G
SHIFT
DATA
SHIFT REGISTER
PARITY CHECK & GEN
PAR
DATA IN BUFFER
G30
G31
G32
G33
G34
G35
G36
G37
G38
G39
G40
G41
G42
G43
G44
G45
G46
G47
G48
G49
G50
G51
G52
G53
G54
G55
29A

DMA CHANNELS

The logic for each Direct Memory Access (DMA) Channel provides 3 distinct functions:

a. indexing the Count/Address (top third of diagram)

b. Transmit control (middle third)

c. unpacking and packing of byte-format data (bottom third)

INDEXING COUNT/ADDRESS

The IODW (I/O Data Word) is delivered to the Channel via the Index Bus In, and is loaded into a backup Buffer by DIL and DIH signals from the Device Controller. DIL (Data Index Low) loads the first 16 bits, and DIH (Data Index High) loads the second 16 bits. IACK (Index Acknowledge) is returned to the Controller on each load.

The loading signal from G30 also sets the Channel Busy flip-flop (so that no other device may request an IODW), and sets the Buffer-Full (BUFFUL) flip-flop. Assuming that this is the first IODW for this transfer, the Index Register is "empty", indicated by the Index-Full (INXFUL) flip-flop being clear. Gate G31 therefore enables G32 to transfer the Buffer contents into the Index Register on the next clock.

Note that G31 also clears the BUFFUL flip-flop (Buffer "empty") which, with the INXFUL signal, enables G35 to send an IWRQ (Index Word Request) to the Device Controller. The Controller accordingly initiates a request for the next IODW, which will proceed if it is holding a Data Chaining bit (bit 0) from the current IODW. The new IODW will be obtained from the I/O Program in memory, and loaded into the backup Buffer during the time that the current block transfer is proceeding.

The COUNT register part of the Index Register is decremented (by the Data Cycle flip-flop) on each word transfer, and the ADDRESS part is incremented. The Address is made available to G48 to be sent to memory, and is applied to the Address Mapper to generate the Destination module number (DES). The Count is applied to G33, which checks for a count of zero; when this occurs, G33 clears the INXFUL flip-flop. Then, depending on whether there is a new IODW waiting in the Buffer, G32 transfers the IODW into the Index Register, or G34 sends a Channel-End (CHEND) signal to the Device Controller.

When the Device Controller signals "Device-End", it disables G30 (no more IODW's) and enables G38. The Controller can then read out the residue contents of the Index Register to the I/O Processor. DIL reads the lower 16 bits onto the Index Bus Out, and DIH reads out the higher 16 bits.

TRANSMIT CONTROL

A high-speed data transmission begins when 4 Channel Requests have been counted. The C=0 signal signifies that a full word has either gone to the device or has been received from the device. This signal sets the Bus Request flip-flop.

The resulting RQB is sent for priority checking to the Port Controller, while LOINH is sent out on the system bus to signify intent to make a LO-priority transmission (address to memory). When the Port Controller responds with Priority Select, G43 sets the Address Cycle flip-flop, if the destination is ready (DESRDY) and module priority allows (LOENB). Address Cycle reads out a memory command word to the system Data bus, consisting of the 20 Address bits at G48 and either a Clear/Write opcode (if Memory-Bound is true) or a Read/Write opcode (if Memory-Bound is false).

Additionally, Address Cycle reads out the TO code (G37), and sets the Data Cycle flip-flop.

If this is a Memory Bound transfer (MEMBND), the Channel has relayed this information (INBND) to the Port Controller. As previously discussed, the Port Controller will request a HI-priority transmission (Set Hi Inhibit flip-flop) to the MCU, which responds with Select when module priority allows. The Select signal at G42 and G44 reads out the TO code and the Data Register In contents to the system Data Bus. G47 clears the Data Cycle flip-flop. Via G51 and G49, Select also sets the Channel Acknowledge flip-flop; CHACK tells the Device Controller to send more data.

If this is a Device Bound transfer (not-MEMBND), G39 waits for a FROM code matching the TO code (DES) sent out to memory in the Address cycle. When this occurs, G46 sets the Hold flip-flop (locking data in the Data Out register), clears the Data Cycle flip-flop, and sets the Channel Acknowledge flip-flop. CHACK (Channel Acknowledge) tells the Device Controller to begin accepting the data.

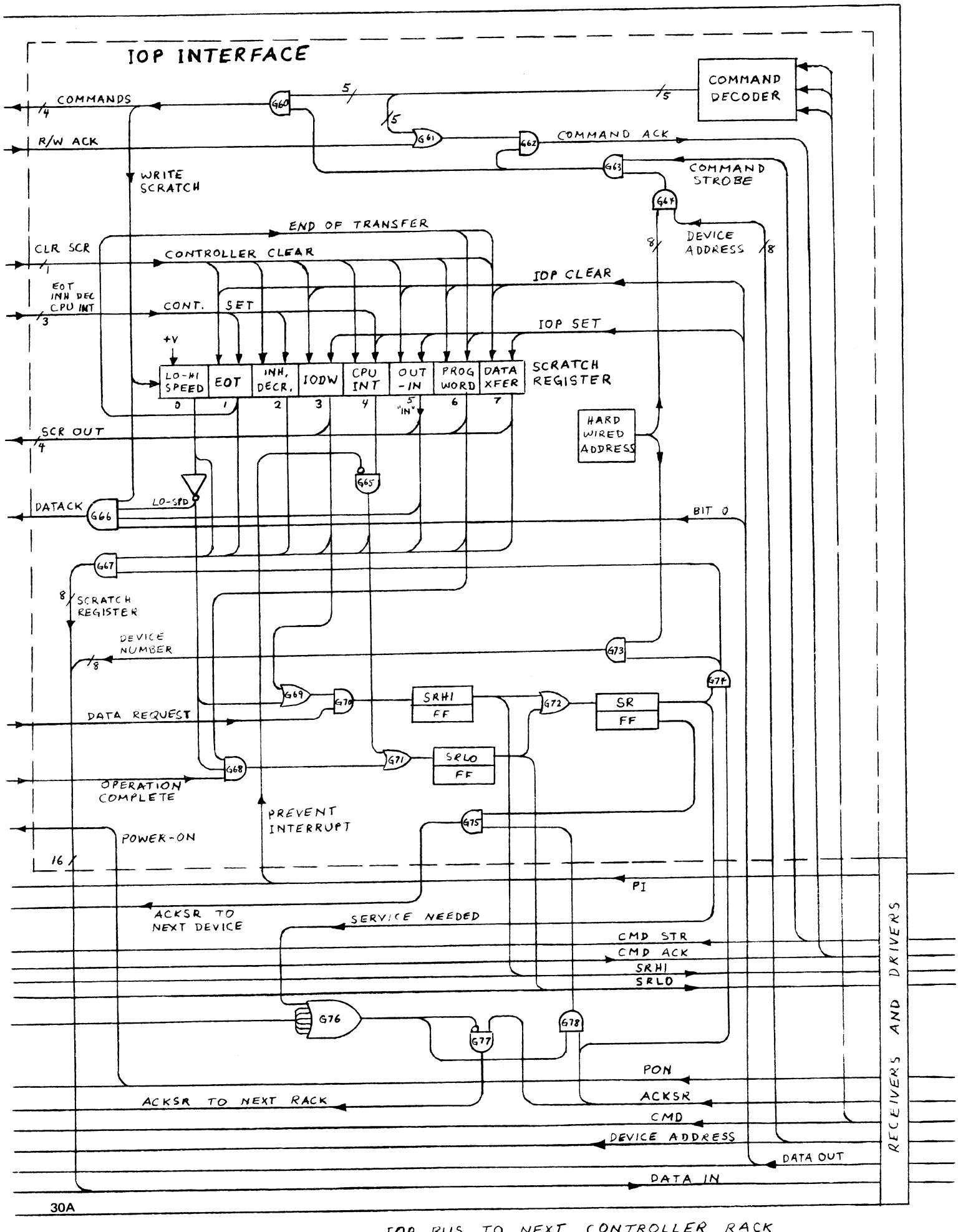
Clearing the Data Cycle flip-flop (when data is sent to or received from memory) decrements the word Count and increments the Address in the Index Register. (Count decrementing can be inhibited by a control line from the Device Controller, not shown here.)

DATA PACKING/UNPACKING

Memory bound data (8 bits) is applied to each input gate, G52, 53, 54, and 55. The accompanying CHRQ, however, is sequentially shifted by a Shift Register so that each gate (representing 8) is enabled in sequence for 4 successive input bytes. These 4 bytes are transferred into the Data Register In for transmission to memory. Parity checking occurs on each input byte, and a Parity Error signal is sent to the Device Controller if an error occurs. Correct parity is independently generated, and accompanies each byte to the Data In Buffer.

Device bound data (36 bits) is received from memory by the Data Out register. Parity is independently generated on each input byte, and compared with each incoming Parity bit. A Parity Error bit, normally false, plus the generated correct Parity bit, shifts to the right, in step with the data. (See next paragraph.)

Each CHRQ produces a Shift signal from the Shift Register, which reads out the rightmost (low order) byte to the device and shifts the remaining bytes to the next right position. After 4 CHRQ's, all 4 bytes and their respective Parity and Parity Error bits have been shifted out.


IOP INTERFACE
COMMAND DECODER
COMMANDS
R/W ACK
WRITE SCRATCH
COMMAND ACK
COMMAND STROBE
DEVICE ADDRESS
END OF TRANSFER
CLR SCR
CONTROLLER CLEAR
IOP CLEAR
EOT INH DEC CPU INT
CONT.
SET
IOP SET
+V
LO-HI SPEED
EOT
INH, DECR.
IODW
CPU INT
OUT -IN
PROG WORD
DATA XFER
SCRATCH REGISTER
0
1
2
3
4
5 "IN"
6
7
HARD WIRED ADDRESS
SCR OUT
DATACK
LO-SPD
BIT 0
SCRATCH REGISTER
DEVICE NUMBER
DATA REQUEST
SRHI FF
SR FF
SRLO FF
OPERATION COMPLETE
PREVENT INTERRUPT
POWER-ON
PI
ACKSR TO NEXT DEVICE
SERVICE NEEDED
CMD STR
CMD ACK
SRHI
SRLO
PON
ACKSR
CMD
DEVICE ADDRESS
DATA OUT
DATA IN
ACKSR TO NEXT RACK
RECEIVERS AND DRIVERS
IOP BUS TO NEXT CONTROLLER RACK

IOP INTERFACE
(Low Speed)

In order to process the commands of an SIO program, each device interfaced to an Omega Device Controller rack requires, in addition to the specific Controller logic, an I/O Processor (IOP) Interface. (It is possible, however, to operate a device under direct control of a CPU, using CIO, RIO, WIO, and TIO instructions. In this case, an IOP Interface may not be required.)

The IOP Interfaces for high and low speed devices are nearly identical (differences will be covered later). The facing diagram illustrates the low-speed IOP Interface.

COMMAND PROCESSING

Commands from the I/O Processor (listed on page 16) are decoded into individual command lines by the Command Decoder (top of diagram). The accompanying Command Strobe, which arrives slightly delayed, enables G63. G64, which determines that the command is for "this device", enables the other input of G63. Then, any true command line at G61 permits G62 to return a Command Acknowledge to the I/O Processor. (The acknowledge for Read and Write commands must come to G61 externally from the Device Controller; this signifies that the commanded action has been taken.

G63 also enables G60 to send the various commands to their respective destinations (one to clock the Scratch Register, and remainder to the Device Controller logic).

SCRATCH REGISTER

If the command is "Write Scratch", information intended to be loaded into the Scratch Register will be present on the Data Out lines of the IOP Bus. As indicated by "IOP Set" and "IOP Clear", the I/O Processor may set bits 3 through 7, and clear bits 1, 3, 5, 6, 7. Bit 0 is permanently wired to indicate "Low-Speed".

The Device Controller may set bits 1, 2, and 4 (individually), and clear (simultaneously) bits 1 through 7.

Bits 3, 5, 6, 7 are available to the Controller, and the entire Scratch Register is read out to the I/O Processor when G67 is enabled by an ACKSR. The significance of the Scratch Register bits is as follows.

Bit	Significance		For
0	Set: Clear:	High speed device Low speed device	Local use
1	Set:	End of Transfer	I/O Processor
2		Not Assigned	
3	Set: Clear:	IODW expected IOCW expected	Controller
4	Set:	CPU Interrupt Requested	I/O Processor
5	Set: Clear:	In Bound Out Bound (to Device)	Controller and Channel
6	Set:	Program Word expected	Controller
7	Set:	Data Transfer requested	Controller

INPUT ACKNOWLEDGE

The I/O Processor tells the device that it has accepted a byte by sending a "1" on bit 0 of the IOP Bus, accompanied by a Write Scratch command. This is not a valid command for the Scratch Register (bit 0 is not connected to that register), so it is used to enable G66 to send a Data Acknowledge (DATACK) to the Device Controller.

PROGRAM WORD REQUEST

The I/O Processor tells the Controller to request a Program Word via bit 6 of the Scratch Register. When the Controller is ready to make such a request, it sends an Operation Complete signal to G68. This sets the SRLO flip-flop (Service Request, LO-priority) through G71. SRLO is sent to the I/O Processor, and sets Service Request (SR) flip-flop while waiting for an acknowledge from the I/O Processor (see ACKSR Propagation below). When this acknowledge arrives, it reads back the Scratch Register contents to the I/O Processor, via G74 and G66. Bit 6 tells the I/O Processor to get the Program Word and send it to the Controller.

DATA REQUEST

Each time the Controller is ready to input or accept byte, it sends a Data Request to G70 which sets the SRHI flip-flop (Service Request, HI-priority). G69 is continuously enabled for low-speed devices; the other input to G69 is inactive. Like SRLO, SRHI is sent to the I/O Processor and sets the SR flip-flop while waiting for ACKSR. ACKSR reads back the Scratch Register to the I/O Processor. Bit 7 says that the Controller wants to transfer a byte, and bit 5 tells which direction, in or out.

CPU INTERRUPT

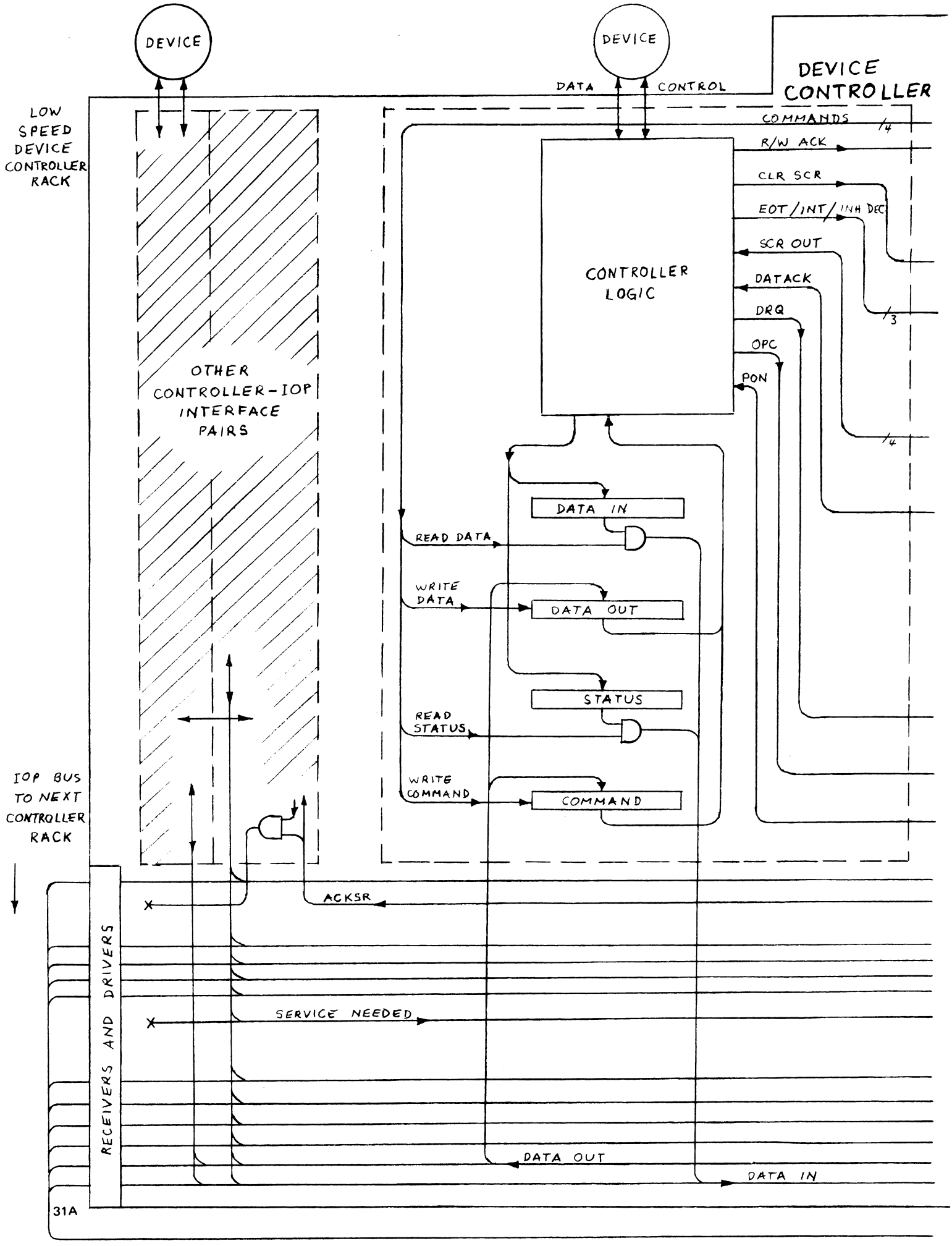
The above two types of requests (Program Word and Data) are initiated by the I/O Processor. A device-initiated transfer may be obtained by means of CPU Interrupt. The I/O Processor may inhibit such requests by a Prevent Interrupt signal to G65. If permitted to occur, the Controller sets bit 4 of the Scratch Register, resulting in an SRLO to the I/O Processor. When ACKSR reads back the Scratch Register, bit 4 tells the I/O Processor to get the Controller's Status Register contents, and send it to a CPU module. It is then up to the CPU to interpret the Status information and, if appropriate, initiate a data transfer.

ACKSR PROPAGATION

ACKSR (Acknowledge Service Request) from the I/O Processor is received at G77 and G78, which are a pair of gates for one Controller rack. G76 applies all Service Requests in that rack to both G77 and G78. If there is no true SR present, ACKSR is passed directly out of the rack via G77.

If there is at least one true SR at G76, ACKSR is routed in series through each IOP Interface in the rack to determine which one is requesting service (or which one has highest priority, if more than one). If the SR flip-flop in the diagram is requesting service (set), G75 is inhibited, preventing propagation of ACKSR any further. G74 is enabled, allowing G73 and G67 to read out the Device Number and Scratch Register respectively.

If the SR flip-flop shown were not set, G75 would pass on the ACKSR to the next IOP Interface in the rack. The first true SR encountered automatically gets highest priority.

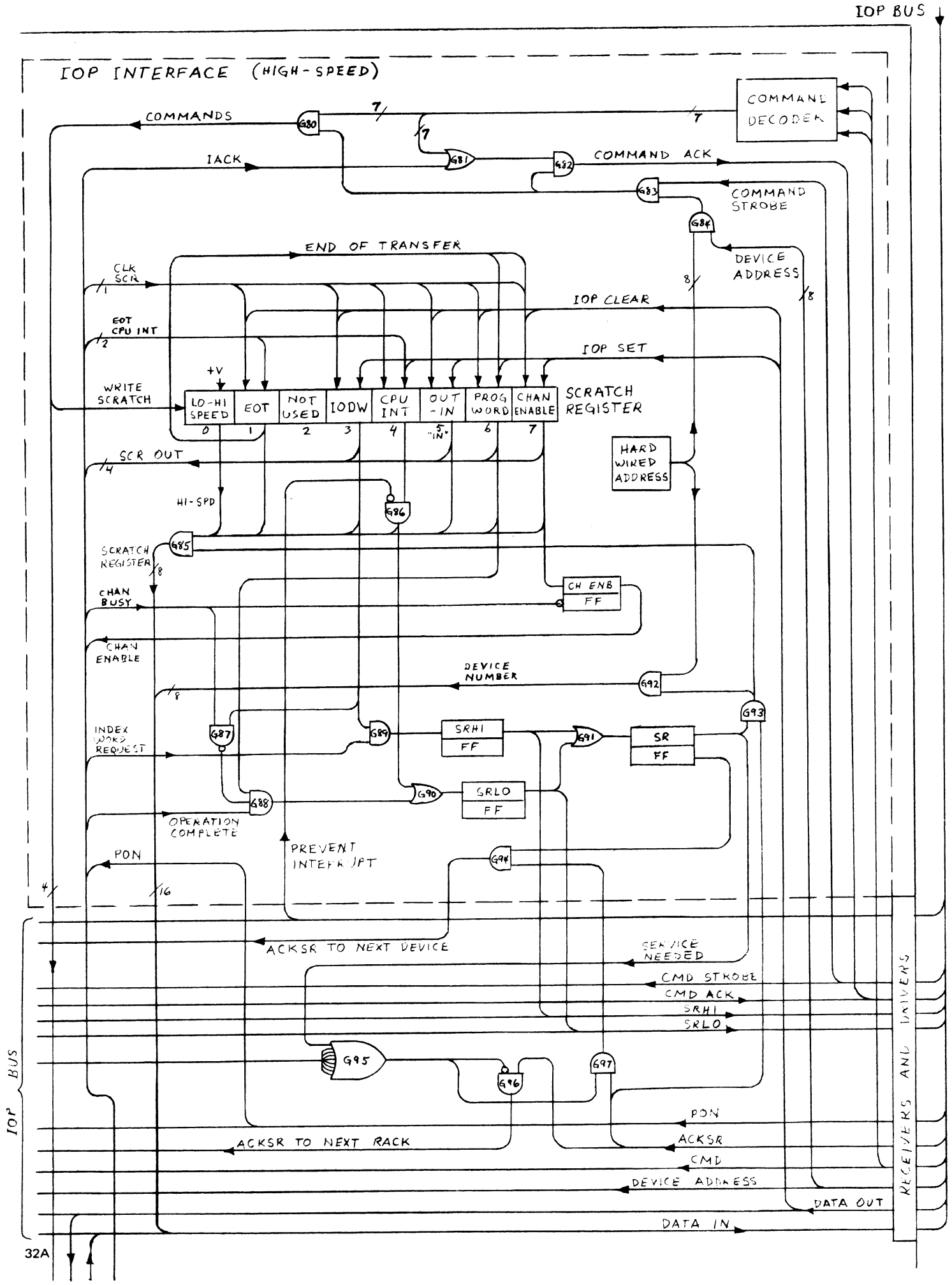

DEVICE
DEVICE
DEVICE CONTROLLER
DATA
CONTROL
LOW SPEED DEVICE CONTROLLER RACK
COMMANDS
1/4
R/W ACK
CLR SCR
EOT/INT/INH DEC
SCR OUT
CONTROLLER LOGIC
DATACK
DRQ
1/3
OPC
PON
1/4
OTHER CONTROLLER-IOP INTERFACE PAIRS
DATA IN
READ DATA
WRITE DATA
DATA OUT
STATUS
READ STATUS
WRITE COMMAND
COMMAND
IOP BUS TO NEXT CONTROLLER RACK
ACKSR
RECEIVERS AND DRIVERS
SERVICE NEEDED
DATA OUT
DATA IN
31A

DEVICE CONTROLLERS

The specific Controller logic for various peripheral devices necessarily varies considerably. Detailed logic, therefore, will not be discussed here. Only the standard control signals and data paths to and from the "Controller Logic" block are shown. The Controller Logic will handle these signals as necessary, perhaps using a ROM Microprocessor, the same as discussed earlier, and translates the signals to specific Control signals for the external device.

In general, however, each Device Controller will have at least one Data In and/or Data Out register, plus a Status Register and a Command Register. The four Command lines allow the Data In and Status Registers to be read out onto the IOP "Data In" Bus, and the Data Out and Command Registers to be loaded from the IOP "Data Out" Bus. Information for the Command Register might have been sent by a CIO instruction from the CPU, or by an IOCW of the Control type from the I/O Program.

The shaded area represents one or more additional pairs of Device Controller and IOP Interface logic units. As shown, the ACKSR which runs in series through each IOP Interface in the rack has no connection to the IOP output connector. "Service Needed" also is a line that is of significance only within the rack, and therefore not connected out. The remaining lines, however, do continue on to the next Controller rack.


IOP BUS
IOP INTERFACE (HIGH-SPEED)
COMMAND DECODER
COMMANDS
G80
7
7
7
G81
G82
COMMAND ACK
IACK
G83
COMMAND STROBE
G84
DEVICE ADDRESS
8
8
END OF TRANSFER
CLK SCR
1
IOP CLEAR
EOT CPU INT
2
IOP SET
+V
WRITE SCRATCH
LO-HI SPEED
EOT
NOT USED
IODW
CPU INT
OUT -IN
PROG WORD
CHAN ENABLE
SCRATCH REGISTER
0
1
2
3
4
5 "IN"
6
7
HARD WIRED ADDRESS
SCR OUT
4
HI-SPD
G86
G85
SCRATCH REGISTER
8
CHAN BUSY
CH ENB FF
CHAN ENABLE
DEVICE NUMBER
G92
8
G93
INDEX WORD REQUEST
G87
G89
SRHI FF
G91
SR FF
G90
SRLO FF
G88
OPERATION COMPLETE
PREVENT INTERRUPT
PON
G94
4
16
ACKSR TO NEXT DEVICE
SERVICE NEEDED
CMD STROBE
CMD ACK
SRHI
SRLO
G95
G97
G96
PON
ACKSR TO NEXT RACK
ACKSR
CMD
DEVICE ADDRESS
DATA OUT
DATA IN
IOP BUS
RECEIVERS AND DRIVERS
32A

HIGH-SPEED IOP INTERFACE

Due to the fact that the DMA Channel, rather than the I/O Processor, handles the actual transfer of data for high-speed devices, there are some minor differences from the low-speed IOP Interface previously discussed. These differences are as follows.

HIGH-SPEED INDICATION

Bit 0 of the Scratch Register is now wired to indicate HI-Speed (output high).

CHANNEL ENABLE

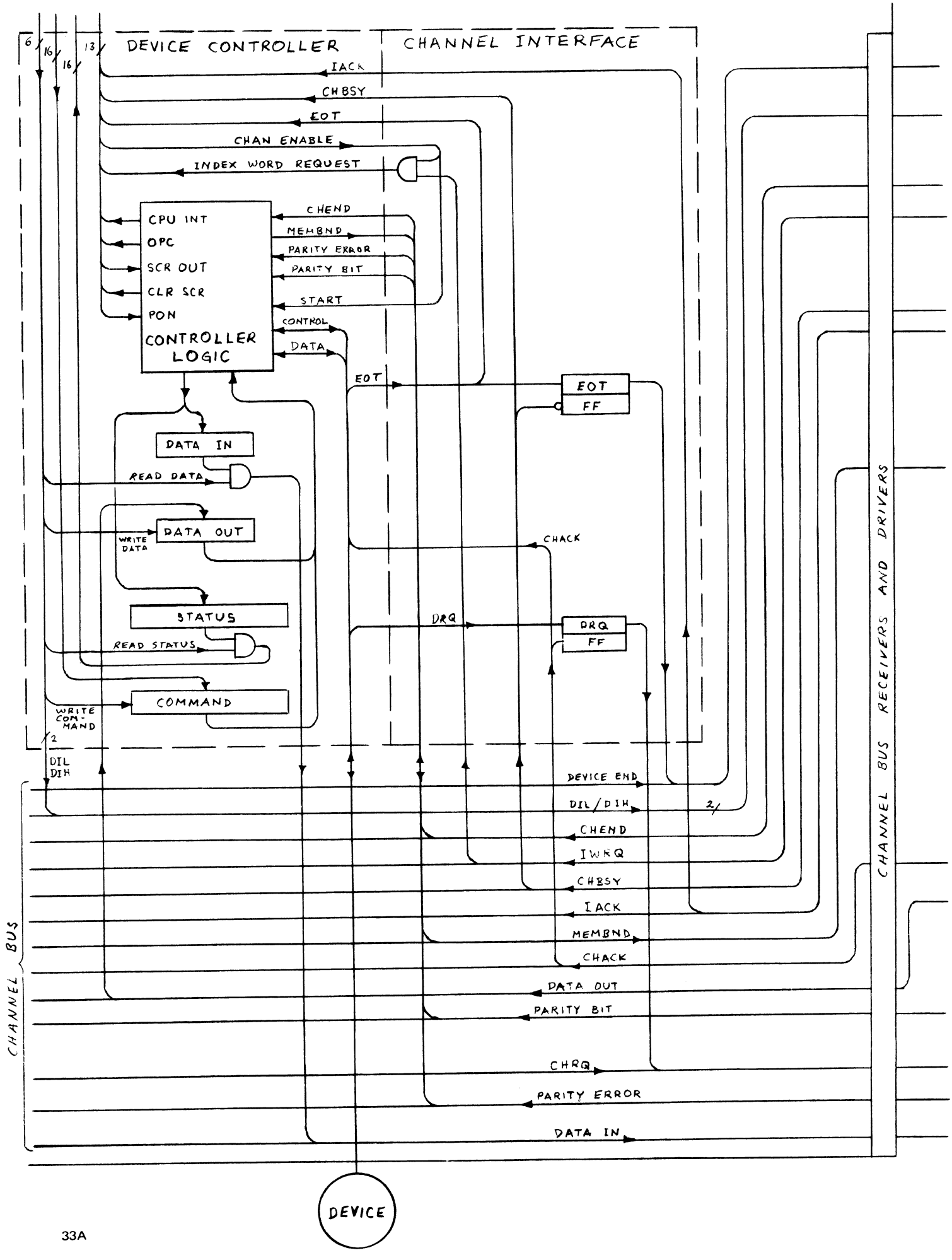
Bit 7 of the Scratch Register (was "Data Transfer" requested) is now Channel Enable. This bit sets a Channel Enable flip-flop, which enables the Channel Interface. The flip-flop is cleared when the Channel finishes its block transfer (last block if data chaining), and goes not-Busy.

INDEX WORD REQUEST

When the Channel needs a new IODW (for data chaining), it sends an Index Word Request to the SRHI flip-flop. If the previous IODW had indicated Data Chaining, bit 3 of the Scratch Register is set, with the result that the Index Word Request initiates an SRHI. The use of HI-priority gives added assurance that the next IODW (for continuing the same transfer) will be available on completion of the current block. Note that the normal request route for IODW's is blocked at G88 by G87. This means that other devices on the Channel may continue to request and process IOCW's up to the point of requesting the Channel for the actual data transfer, but may not request the IODW. This contributes significantly to overall system speed, owing to the fact that preliminary mechanical actions of other devices may be occurring (under control of IOCW's), while the Channel is busy transferring the data of the current device.

DIL/DIH COMMANDS

Two additional Commands are effective: "Issue Data Index Low" (DIL) and "Issue Data Index High" (DIH) to Channel. (See page 16 for complete list of Commands.) When the Controller has received these Commands from the IOP Interface, it returns IACK (Index Acknowledge), which the IOP Interface relays to the I/O Processor as "Command Acknowledge". (Read/Write Acknowledge is no longer present.)


DEVICE CONTROLLER
CHANNEL INTERFACE
6
16
16
13
IACK
CHBSY
EOT
CHAN ENABLE
INDEX WORD REQUEST
CPU INT
OPC
SCR OUT
CLR SCR
PON
CONTROLLER LOGIC
CHEND
MEMBND
PARITY ERROR
PARITY BIT
START
CONTROL
DATA
EOT
EOT
FF
DATA IN
READ DATA
DATA OUT
WRITE DATA
CHACK
STATUS
DRQ
DRQ
FF
READ STATUS
COMMAND
WRITE COM-MAND
2
DIL
DIH
DEVICE END
DIL/DIH
2
CHEND
IWRQ
CHBSY
IACK
MEMBND
CHACK
DATA OUT
PARITY BIT
CHRQ
PARITY ERROR
DATA IN
CHANNEL BUS
CHANNEL BUS RECEIVERS AND DRIVERS
DEVICE

CHANNEL INTERFACE

The Channel Interface provides logic for servicing Data Requests (DRQ), End of Transfer (EOT), and Index Word Requests (IWRQ). Since the logic itself is simple (two flip-flops and a gate, as shown), the following discussion is expanded to review the entire sequence of a high speed transfer. Some reference to the Channel and IOP Interface (High-Speed) diagrams will be necessary; these diagrams are intended to adjoin the facing diagram at right and top, respectively.

COMMAND THE DEVICE

Before the transfer starts, a device may require preliminary start-up commands. These are applied in the usual way (IOCW or CIO instruction) from the I/O Processor, via the IOP Bus, to the Command Register. The Controller Logic translates the Command Register contents to specific control signals to the device.

INITIALIZE

The first step involved in a transfer is for the I/O Processor to tell the Channel to load the IODW. The I/O Processor does this by putting out the lower 16 bits of the IODW on the IOP Bus and addressing a "send DIL" command to the device (IOP Interface), not directly to the Channel. The IOP Interface accordingly sends DIL (Data Index Low) to G30 in the Channel, via the Channel Bus. (Routing DIL/DIH commands through the Device Controller makes it unnecessary for the I/O Processor, or the programmer, to know which Channel a given device is connected to.) Since we are assuming the Channel is presently not-Busy, the EOT flip-flop in the Channel Interface is clear; this means Device End is low, enabling G30 to load the waiting IODW bits into the Channel's Buffer.

Now the Channel goes Busy; CHBSY sent to G87 in the IOP Interface prevents other devices on the Channel from requesting an IODW. Also, IACK (Index Acknowledge) is returned from the Channel to the I/O Processor as a Command Acknowledge. The I/O Processor then puts out the higher 16 bits of the IODW, and commands the device (IOP Interface) to send DIH to the Channel. DIH (Data Index High) loads the high IODW bits into the Channel's Buffer.

On the next clock, the Channel transfers the IODW from the Buffer to the Index Register. At this point, if this is an output (device bound) operation, G41 initiates a Bus Request in order to obtain the first word from memory while the following operations are taking place. The Buffer-empty and Index-full indications also enable G35 to send IWRQ (Index Word Request) to the Channel Interface, where it waits for Channel Enable. IACK returned to the I/O Processor allows it to send Channel Enable (bit 7 of the Scratch Register). The IWRQ is now forwarded to G89 in the IOP Interface. If the I/O Processor has detected a Data Chaining bit in the current IODW, it has set bit 3 of the Scratch Register, thus allowing an SRHI for the next IODW to proceed. The IODW will arrive at the I/O Processor at a random time, and will be loaded into the Channel's Buffer.

Channel Enable also enables the Controller Logic, and issues a Start signal to the device.

TRANSFER

When the device has responded to the Start command (has read input data or prepared for output data), it issues Data Request, DRQ. If this is an input operation, the Controller Logic loads the input byte into the Data In register when it gets DRQ. Setting the DRQ flip-flop sends Channel Request CHRQ to the Channel, thus beginning the input or output operation.

If input, CHRQ enables G55 to load Byte 3 and its parity bit. The G55 strobe also sets the Channel Acknowledge flip-flop via G49. CHACK clears the DRQ flip-flop and causes the Controller to send an acknowledge the device. This allows the device to send another byte, with DRQ. DRQ sends CHRQ, which is shifted to enable G54 to load Byte 2. The process repeats with G53 and G52 loading Bytes 1 and 0. Then the four bytes are transferred into the Data In Buffer, and the Shift Register issues C=0 (Count = 0) to initiate a transmission to memory. (RQB, Priority Select, Address Cycle, Data Cycle.) The Data Cycle's Select causes the fourth CHACK, via G51, allowing the device to get more data while the memory transmission is occurring.

If output, the memory data is transferred from the Data Out Buffer to a shift register, and CHRQ causes a Shift signal that sets Channel Acknowledge. C=0 causes a Bus Request, to reload the Data Out Buffer while the Channel is outputting the current data as follows. CHACK reads out Byte 0, via G50, and causes the Controller Logic to load the byte into its Data Out register. CHACK also clears the DRQ flip-flop, and tells the Controller Logic to send an acknowledge to the device, along with the data. (Meanwhile, the Shift signal moves Bytes 1, 2, and 3 to the left.) When the device has accepted the first byte, it sends another DRQ. The resulting CHRQ regenerates Shift, which sets Channel Acknowledge. CHACK reads out Byte 1 via G50, and causes the device to accept the byte. This process repeats until all four bytes have been read out. The fifth CHRQ causes C=0 to transfer the next word from the Data Out Buffer, and to initiate a Bus Request to get another word for the Buffer.

END OF TRANSFER

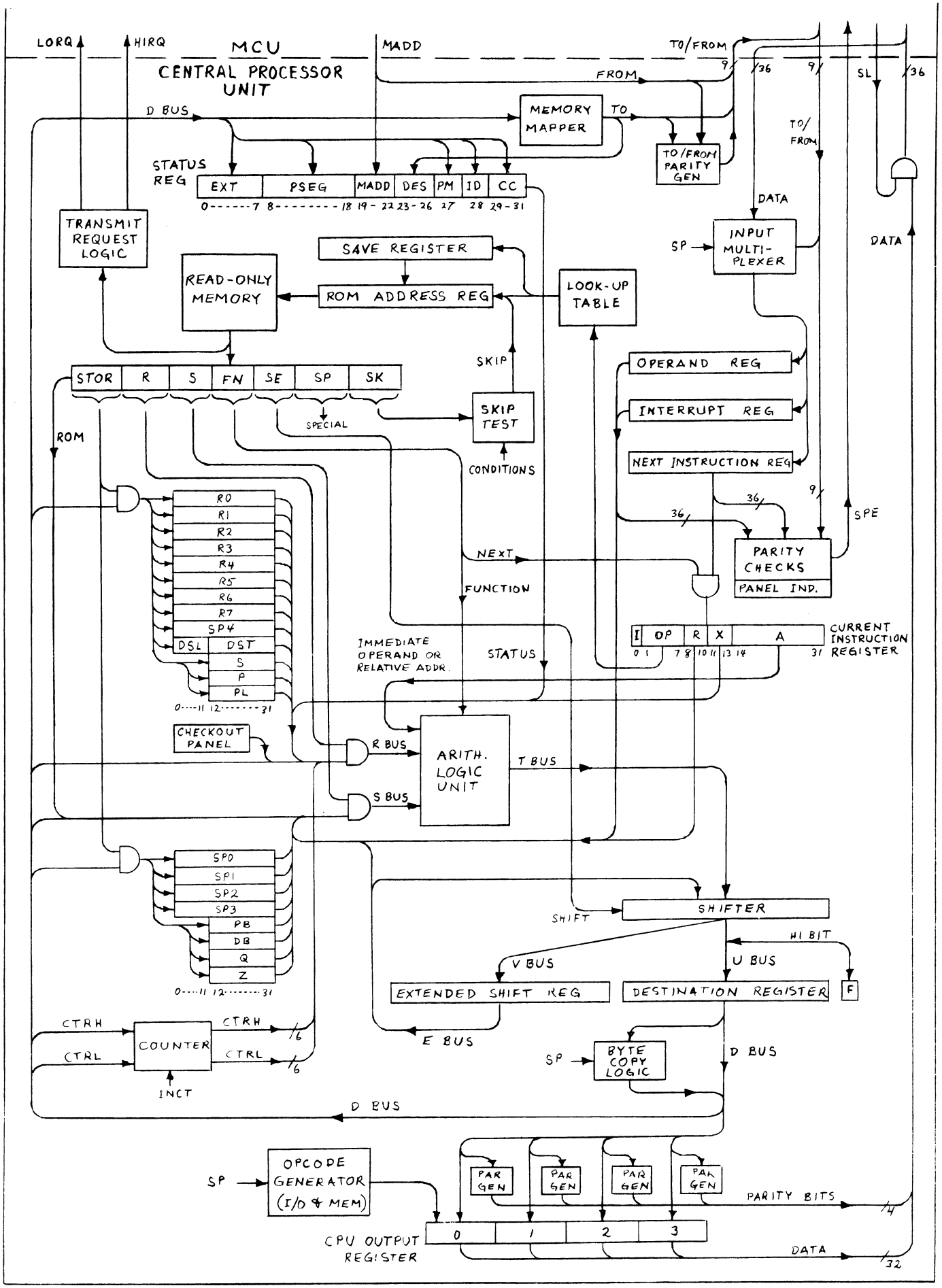
If large blocks are being transferred, the COUNT in the Index Register may go to zero before completion of the transfer. In this case, G33 clears the Index-Full flip-flop (INXFUL), causing G31 to load the Buffer contents into the Index Register, and G35 to send out for another IODW (IWRQ signal; see Initialize). Data transfer can thus continue.

When the last byte of data has been transferred, the device issues Operation Complete (OPC), and sets the EOT (End-of-Transfer) flip-flop. EOT sets up the Channel to read out the residue of the Index Register, by sending Device End to G38. Meanwhile, EOT also sets the EOT bit in the Scratch Register, which in turn sets the Program Word request bit, and clears the Channel Enable bit.

Since there is no further IODW request from the I/O Processor, bit 3 of the Scratch Register is clear, thus allowing G87 to enable G88. Operation Complete therefore is allowed to initiate an SRLO to the I/O Processor. The I/O Processor, when ready, reads back the Scratch Register, and in recognition of the EOT bit, causes a DIL (followed by DIH, after IACK). With Device End at G38, DIL/DIH now reads the Index residue back to the I/O Processor.

At the same time, G36 clears the Channel Busy flip-flop. This in turn clears the Channel Enable flip-flop, and allows G87 in the other device controllers to make an SRLO for their IODW's. (The highest priority SRLO will be acknowledged.)

If the amount of data transferred is greater than the allotted Count, the condition will occur in which Buffer-Full and Index-Full will be clear before End-of-Transfer occurs. G34 will then send a Channel-End (CHEND) signal to the Device Controller, which interprets CHEND as an alarm condition. The Controller may then take appropriate action, such as setting a Status bit and interrupting a CPU.


MCU
CENTRAL PROCESSOR UNIT
LORQ
HIRQ
MADD
TO/FROM
FROM
SL
D BUS
MEMORY MAPPER
TO
TO/FROM PARITY GEN
STATUS REG
EXT
PSEG
MADD
DES
PM
ID
CC
0------7 8--------18 19-22 23-26 27 28 29-31
TRANSMIT REQUEST LOGIC
SAVE REGISTER
READ-ONLY MEMORY
ROM ADDRESS REG
LOOK-UP TABLE
DATA
SP
INPUT MULTI-PLEXER
STOR
R
S
FN
SE
SP
SK
SKIP
SPECIAL
SKIP TEST
CONDITIONS
ROM
OPERAND REG
INTERRUPT REG
NEXT INSTRUCTION REG
SPE
PARITY CHECKS
PANEL IND.
NEXT
FUNCTION
R0
R1
R2
R3
R4
R5
R6
R7
SP4
DSL
DST
S
P
PL
0----11 12-------31
IMMEDIATE OPERAND OR RELATIVE ADDR.
STATUS
I
OP
R
X
A
CURRENT INSTRUCTION REGISTER
0 1
7 8
10 11
13 14
31
CHECKOUT PANEL
R BUS
S BUS
ARITH. LOGIC UNIT
T BUS
SP0
SP1
SP2
SP3
PB
DB
Q
Z
SHIFT
SHIFTER
HI BIT
V BUS
U BUS
EXTENDED SHIFT REG
DESTINATION REGISTER
F
E BUS
CTRH
CTRL
COUNTER
INCT
BYTE COPY LOGIC
D BUS
OPCODE GENERATOR (I/O & MEM)
PAR GEN
PARITY BITS
CPU OUTPUT REGISTER
0
1
2
3
DATA
32

CENTRAL PROCESSOR MODULE

CENTRAL PROCESSOR LOGIC.

The following description of Central Processor logic assumes familiarity with the Omega system architecture, described in a separate document. Only the basic logic is discussed here, based on the sequence of events involved in executing an instruction.

First note the major logic elements shown in the facing diagram. (The Module Control Unit, which together with the CPU proper comprises the Central Processor Module, is omitted from the top of the diagram since it was detailed earlier in this manual.) Counterclockwise from top left: the Read-Only Memory is addressed by the ROM Address Register, and outputs its microprogrammed instructions to the ROM Output Register (in addition to controlling the Transmit Request Logic). The seven fields of the ROM Output Register (4 or 5 bits each) control all of the internal CPU operations. The STOR field selects one of several registers (22 shown in this diagram) in which to store the D bus. The R field selects one of several sources (19 shown, including a Checkout Panel), to read onto the R bus, and the S field selects a source (14 shown) for the S bus. The FN (Function) field selects various ALU functions, plus other functions not indicated in the diagram. The SE (Shift Enable) field controls manipulation of T bus bits in the Shifter, before storage of bits in the Destination Register. The SP (Special) field performs miscellaneous controls throughout the CPU (only 4 shown here, indicated by SP and INCT), and SK (Skip) selects one of various bit or register conditions for skip testing (Condition Code Flag, Overflow, register odd/even/positive/negative, etc.).

In the two arrays of registers shown as R and S bus inputs, the 17 registers excluding the SP registers are the only ones available to the programmer. The remaining five SP (Scratch Pad) registers are for use by the ROM program only. The same is true of the Counter (CTR, lower left corner), which provides a 6-bit count register for use by the ROM program; the two sets of inputs and outputs are for normal counting or floating point usage.

The CPU Output Register (bottom) receives the D bus information from the Destination Register. Its contents, with Parity generated for each 8 bits, can be selected (SL) by the MCU for transmission to other modules. The Opcode Generator, under control of the SP field, inserts appropriate opcodes when sending command words to I/O or memory. The Byte Copy Logic is used only when transmitting individual bytes; to be sure the byte is in the correct position in the word, the byte is simply copied into all four positions before transmitting. (The receiving module must know, or decode from the Opcode, which byte position is valid.)

The shift logic includes the Shifter (a series of gates, controlled by the SE field), an Extended Shift Register, and a Flag bit, in addition to the Destination Register, which usually holds the end result of shifting. Two independent sets of input and output buses (U,V,D,E) provide a wide range of shift possibilities.

The Current Instruction Register holds the instruction currently being executed. It is loaded (gated by NEXT signal) from the Next Instruction Register, which in turn received its contents from memory. To save time, the ROM program will send a request to memory for the next instruction as soon as the NEXT transfer occurs. The other two inputs from the MCU are the Operand Register (for expected responses from other modules) and the Interrupt Register (for unsolicited inputs). The presence of an Interrupt is not tested until the ROM program is in a position to process the Interrupt. Parity is checked on all three types of inputs (after reading out of the register), and also on the incoming TO/FROM bits; a parity error triggers a parity error trap routine, sets a panel indication, and sends out a System Parity Error signal. The Input Multiplexer decides where to route incoming information from the MCU.

The Status Register is loaded primarily by the ROM program (from the D bus), but also provides Module Address (MADD), from the MCU) and Destination (from the Memory Mapper). Parity is generated on the 8 TO/FROM bits when transmitting.

SEQUENCE OF EVENTS

Assume that the CPU is currently executing a program and that the current instruction is not of an addressing-modifying type (jump, skip, etc.).

1. The ROM program calculates and loads an absolute memory address into the CPU Output Register, together with a Read/Write Opcode, then telis Transmit Request Logic to issue a LORQ. The destination module is as mapped from the address by the Address Mapper.
2. The MCU, when clear to transmit, issues Select (SL) which sends the Address/Opcode to memory. The Input Multiplexer, expecting a response from that module, routes the returned instruction word to the Next Instruction Register.
3. When the current instruction has been completed, the NEXT signal from ROM loads the new instruction into the Current Instruction Register.
4. The OP field of the instruction causes the Look-Up Table to look up the starting address of the ROM microprogram for that instruction.
5. The starting address is loaded into both ROM address registers, and they step together through the microprogram. (The Save Register stores the present address when the microprogram does a JSB.)
6. ROM reads out the addressed microinstruction to the ROM Output Register. Through logic not shown, the various fields of the instruction word send out control signals to all CPU logic elements, as described above, dependent on the specific instruction.
7. The first action is to trigger a request to refill the Next Instruction Register. The P Register contents (Program Counter) is sent via the R bus to the ALU, which increments the value, and (besides storing back in P) sends it out to program memory as in steps 1 and 2 above.
8. Meanwhile, assuming the instruction requires an operand, the ROM microprogram reads the A field of the instruction to the ALU for address computation with DB, Q, and/or S on the R and S buses.
9. The resultant address (and Memory Opcode) is sent out to memory, while ROM waits for the return transmission of the operand. (If there is another operation ROM could be doing, it may do so at this point.)
10. The returned operand is routed by the Input Multiplexer to the Operand Register, and the read-in strobe also lets ROM know that it is ready to be processed.
11. The Operand is read onto the S bus, with possibly one of the R bus register inputs, for manipulation in ALU.
12. When the instruction is fully executed, the result is disposed of as required (perhaps to memory, or a register), and a new NEXT signal repeats the process (to step 3).