1

Neptune Memory System

1.1 Functional Description

1.1.1 System Overview

The C3 memory system links up to nine Processor-I/O ports to up to eight memory boards and one communications register board (commreg). Each of the Processor-I/O ports can be either a cpu or an I/O port- the interface is identical as far as the memory system is concerned. Hereafter, these ports will be referred to simply as processor ports.

1.1.1.1 C2 versus C3

The following is a discussion of the major differences between the C2 and C3 memory systems.

- Processor to memory interconnect was a single bus driven from the processor to each of the four memory boards on a side (even or odd) of memory. All C3 busses are to be single source, single destination to avoid degradation of signals on multi-tap busses.

- On the C2, the crossbar function of switching the processor requests to and from individual memory banks was performed partly by nature of the multiple destination busses and partly within the crossbar gate arrays on each memory board. The crossbar was thus distributed across each of the memory boards and each board had five ports- one for each of the processors and I/O. The C3 system will have a single, central crossbar, separate functionally and physically from the memory boards. Each processor has a single path to the crossbar and each memory board has a single path to the crossbar.

- The C2 memory boards contained eight banks of memory, 32 bits in width. Two boards were necessary to provide one longword of data per clock. The C3 memory boards contain 32 banks organized as two 32 bit wide sets of sixteen banks each. Only a single C3 memory is therefore required per processor as compared to two C2 memory boards per processor. Eight memory boards were required to support the C2's four processors. Eight C3 memory boards are necessary for the C3's eight processors.

- Each memory bank on a C2 memory board had its own gate array to perform Error Detect and Correct (EDC) and to generate the individual bank control signals. The C3 memory boards will have a single gate array for preforming EDC for reads, four bank control gate arrays, each controlling eight banks, and one gate array per bank for write EDC.

- The C2 memory system can utilize 256K, 1M and 4M RAMs. The C3 system can use 1M and 4M RAMs only. Minimum memory configuration (per board pair) for the C2 was 16M (256K RAMs, single row), maximum configuration, with MCM2s and 4M RAMs, was 1G. A C3 board may have from 128M to 512M. No row expansion is possible on C3 memory boards.

1.1.1.2 Memory Boards

Each of the memory boards in the memory system is capable of returning one 64 bit result per clock and is therefore capable of supporting one processor making its maximum number of requests (one per clock). At a clock period of 16ns, this gives a 500MB/sec bandwidth for a single board and 4GB/sec bandwidth for a full system of eight boards.

The 64 bits of data that a memory board can handle is divided into two completely independent words, called even and odd. Each "side" is separately addressable with separate send and return datapaths. Independent even and odd words allows any word or any word aligned longword to be accessed in a single clock.

The memory boards are to be designed to accept one and four megabit dynamic RAMs. With 1M RAMs, each memory board will hold 128MB of memory; 4M RAMs yield 512MB. A full system of eight boards could therefore have 1G to 4G of memory, depending on RAMs used.

1.1.1.3 Interleaving

The RAMs to be used in the memory system have a cycle time on the order of fifteen system clocks. In order for a single board to support a single processor making requests at the rate of (up to) one request per clock, it is necessary to have sixteen independent banks. With the independent even/odd word organization of a memory board, a total of thirty-two word banks are required per board.

Table 1-1: Memory Board Addresses

# Boards	RAM	Bd Sel	Bank Sel	ADDR
1	1M	<31:30:29>	<6..3>	<26..7>
2	1M	<31:30:7>	<6..3>	<26..8:29>
4	1M	<31:8:7>	<6..3>	<26..9:30:29>
8	1M	<9:8:7>	<6..3>	<26..10:31:30:29>
1	4M	<31:30:29>	<6..3>	<28..7>
2	4M	<31:30:7>	<6..3>	<28..8:29>
4	4M	<31:8:7>	<6..3>	<28..9:30:29>
8	4M	<9:8:7>	<6..3>	<28..10:31:30:29>

Low order address bits are used for bank select so that sequential accesses to a single board hit different banks instead of hitting an already busy bank. In systems with more than one memory board, board select is also generated from low order address bits, also to improve sequential accesses. Table 1-1 shows how bank and board select is generated from a physical address. The processors will swap bits from the <9..7> range into board selects and replace them with bits <31..29> to provide low order interleaving.

It is possible to operate the system with three, five, six or seven boards, or to mix 1Meg and 4Meg boards (however all the RAMs on a particular board must be of the same type). In these cases, interleaving is only possible across the largest power of two group of boards of the same memory type, any remaining memory boards that do not fit into this power of two group will have a lower level of interleaving. Thus, with five memory boards, each 1M, the first four boards could be interleaved across for an effective interleave of 64 way, longword but the fifth board could not be interleaved with any other board and so could only operate at its base 16-way longword interleave.

To provide the bandwidth necessary to support vector operations on configurations with three, five, six or seven processors, it is necessary to have a number of memory boards equal to *2 raised to the ceiling(log(# processors))* where the *log* function is base two.

1.1.1.4 Crossbar

The nine processor ports and eight memory boards plus the commreg board are connected by a central assembly called the crossbar. Each processor communicates to the crossbar on a point to point bus. Multi-drop busses are not used. The same applies to crossbar/memory board interfaces.

The crossbar routes data to and from the memory boards and arbitrates between requests to identical memory boards. The memory boards themselves simply accept one even and one odd request and return one even and one odd request per clock. The crossbar determines which processor is allowed access to the board on a clock and where any return data is to go.

1.1.2 Memory Boards

1.1.2.1 Organization

Figure 1-1 is a high level block diagram of a C3 memory board showing both even and odd sides of the board (even side at the top, odd side at the bottom of the figure). A Neptune memory board is organized as two independent 32 bit sides of sixteen banks each. One side of this board is addressed as even memory (address bit <2> equal to 0), the other as odd memory (address bit <2> equal to 1). (Together, a memory board can return any word aligned longword per clock which is sufficient to access any byte, halfword or word, or any word aligned longword in a single clock.)

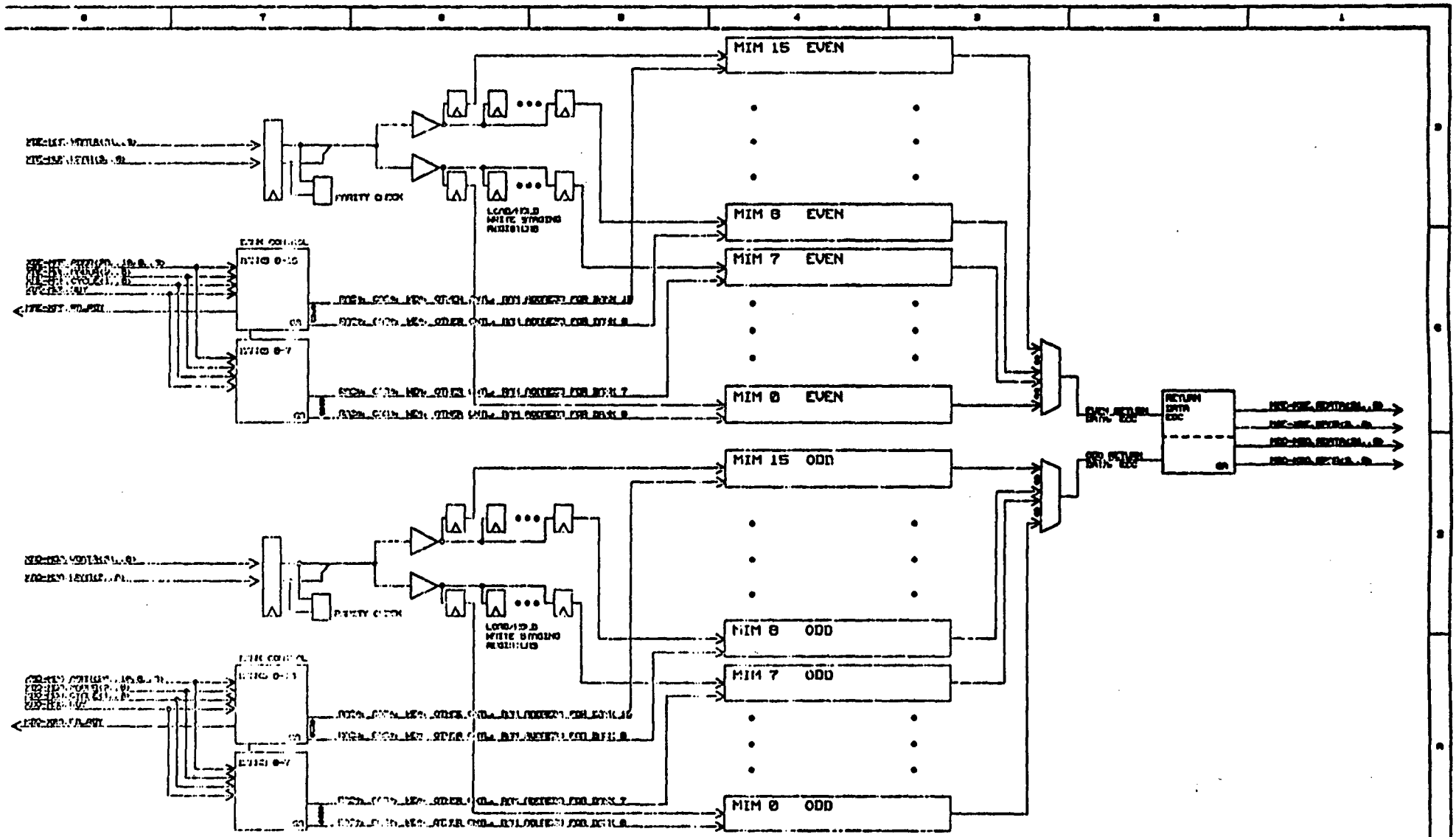
A memory board consists of the motherboard itself which has all the bank control, some of the EDC, all of the ECL staging and multiplexing, and thirty-two MIMs (Memory Inline Module). The MIMs are small cards mounted on the motherboard. Each card holds the necessary 39 RAMs (32 for data, 7 for ECC) required for a single bank, all the ECL-to-TTL and TTL-to-ECL translators, and a small CMOS array for generating write ECC on writes and read-modify-writes.

Unlike the C2 memory system, there is no room on the MIM for row expansion. A single bank therefore consists of a single row of RAMs, all of which are active any time the bank is in use. A single MIM will hold 32 bits of data and 7 bits of ECC. With 1M RAMs, this corresponds to 4MB of data per MIM, or 128MB for all 32 banks. With 4M RAMs, there are 16MB per MIM and 512MB for the entire memory board.

A memory board with its MIMs contains three types of gate arrays. On the motherboard, there are four bank control gate arrays (BCGA) and one read EDC gate array (RDEDC). The third gate array type is the write EDC gate array (WREDC) found on each MIM. The BCGA control eight banks, generating address and control strobes for each bank. The RDEDC gate array performs the EDC for return data. The single RDEDC part handles return data for both even and odd sides. The WREDCs perform parity check and ECC generate for writes and partial writes to a particular bank. Table 1-2 gives the number and location of these gate array types for a single memory board populated with its thirty-two MIMs.

1.1.2.2 Interface

The memory board crossbar interface is given in Table 1-3 . Each memory board has a single even side and a single odd side interface to the crossbar. It interfaces to no other boards. As with the crossbar, signal timing is considered to be register to register with no intermediate levels of logic.



CONVEX

TITLE: MAB OVERVIEW
DRAWING: C30-000000-03xx A
REVISED: Wed Aug 31 18:48:40 1988
ABBR: MAB
ENGR: GUATTERMANN
PAGE: A OF A

Table 1-2: Memory Board/MIM Gate Array Summary

Gate Array	Size	Total Number	Unit	Location
Bank Control	20K	4	One per eight banks	Motherboard
Read EDC	10K	1	One for both even and odd return data	Motherboard
Write EDC	4K	32	One per bank	MIM

Table 1-3: Crossbar to Memory Board I/O

Signal	Description	Width
XBE-MxE.RDY	Crossbar has even board request	1
XBE-MxE.ADDR<28..3>	Request address	26
XBE-MxE.WR_ZONE<3..0>	Request write zone	4
XBE-MxE.WR_DATA<31..0>	Request write data	32
XBE-MxE.WR_PAR<3..0>	Request write data parity	4
XBE-MxE.CYCLE<1..0>	Request operation type	2
XBE-MxE.CTL_PAR<4..0>	Request parity for control	5
MxE-XBE.RD_DATA<31..0>	Return data	32
MxE-XBE.RD_PAR<3..0>	Return data parity	4
MxE-XBE.RD_RDY	Return data available	1
MxE-XBE.DONE<15..0>	Finished signal for each bank	16
XBO-MxO.RDY	Crossbar has odd board request	1
XBO-MxO.ADDR<28..3>	Request address	26
XBO-MxO.WR_ZONE<3..0>	Request write zone	4
XBO-MxO.WR_DATA<31..0>	Request write data	32
XBO-MxO.WR_PAR<3..0>	Request write data parity	4
XBO-MxO.CYCLE<1..0>	Request operation type	2
XBO-MxO.CTL_PAR<4..0>	Request parity for control	5
MxO-XBO.RD_DATA<31..0>	Return data	32
MxO-XBO.RD_PAR<3..0>	Return data parity	4
MxO-XBO.RD_RDY	Return data available	1
MxO-XBO.DONE<15..0>	Finished signal for each bank	16

The memory board registers the request signals from the crossbar each clock. If the RDY signal is true, it will farm out the request to the bank controller for the requested bank. Since the arbitration logic on the crossbar keeps track of which banks are busy and the individual banks indicate when they are no longer busy (via DONE), it is not possible for a request to arrive at a busy bank.

Return data is always available a fixed number of clocks after a read or a test-and-modify operation is requested. Since return time is fixed and only a single request can be made on each clock (on each of the independent even and odd sides of the board), only one piece of data on each side can be ready on each clock. Each clock new return data is placed on the RDATA bus. If no data is expected on the current clock, the previous clock's data will be driven, ensuring good parity.

The memory board asserts RD_RDY with each piece of return data but this signal is only used for debug and error checking. The return control in the crossbar has no functional need for this signal. Given the number of programmable registers necessary in the memory system to allow for various speed DRAMs, this error checking will make it easier to detect mismatches in the times programmed for the crossbar and memory boards.

As with the memory boards, the crossbar consists of two 32 bit wide, independent sections. Together, the two crossbar sections can handle 64 bits of data per clock per processor port in both the store and return directions for a total of 1152 bits of data per clock. Each 32 bit crossbar is completely independent of the other. Each receives its own set of addresses and data and talks only to the even or only to the odd port of the memory boards.

Figure 1-2 depicts the functional blocks of a single, 32 bit side of the crossbar. Both even and odd sides are identical with no communications between the two sides.

The crossbar is composed of four gate array types. The ARB gate array determines which processor gets access to a particular memory board if there is contention for a board. It also controls the input queues on the SEND_XBAR gate arrays. Each ARB can control the access to four memory boards and one commreg board, therefore two are required per crossbar side. (Only one of commreg ports is used.)

The SEND_XBAR consists of input queues for processor data and a multiplexor to select data from these queues for each of the memory ports. The selects for these multiplexors are controlled by the ARB gate arrays as are the controls for these queues.

The RTNCTL gate array notes when a read request is issued to a memory board so that it can generate the proper multiplexor selects and a read ready (RD_RDY) for that data at the proper time. The RTN_XBAR gate arrays, like the SEND_XBAR gate arrays, simply multiplex data. They select one of the eight memory boards or the commreg board data for each of the processor return data ports.

Table 1-4: Crossbar Gate Array Summary

Gate Array	Size	Total Number	Unit
ARB	20K	4	One per four memory boards per side
SEND_XBAR	10K	22	One per 7 bits of send data
RTNCTL	20K	2	One for each side's return data
RTN_XBAR	10K	10	One per 8 bits of return data

Table 1-4 gives a summary of the types and estimated numbers of gate arrays in a complete crossbar (both even and odd sides).

1.1.3.2 Processor Interface

A processor makes a request to the crossbar by asserting the RDY signal to the crossbar. If REQ_NEXT was asserted on the *previous* system clock, the crossbar will accept the request-taking in the address, board select, data, etc and performing the appropriate operations.

The RDY and REQ_NEXT signals, as well as all the other processor-crossbar signals, are all to be sourced directly from a register clocked at system clock rising. At the destination all signals can only be assumed to make set-up to a similar register. Both source and destination registers can be within gate arrays. Therefore all processor-crossbar communication is to be register to register. Physical layout of the processors and crossbar, given the system clock period, allows for no intermediate levels of logic.

Table 1-5: Crossbar to Processor I/O

Signal	Description	Width
PxE-XBE.RDY	Processor has a valid even request	1
XBE-PxE.REQ_NEXT	Crossbar can accept req next clock	1
PxE-XBE.ADDR<28..3>	Processor request address	26
PxE-XBE.BD_SEL<3..0>	Processor board select	4
PxE-XBE.WR_DATA<31..0>	Processor write data	32
PxE-XBE.WR_PAR<3..0>	Processor write data parity	4
PxE-XBE.WR_ZONE<3..0>	Processor write zones	4
PxE-XBE.CYCLE<1..0>	Processor operation code	2
PxE-XBE.CTL_PAR<4..0>	Parity for Address and Control	5
XBE-PxE.RD_RDY	Return data ready for processor	1
XBE-PxE.RD_DATA<31..0>	Return data for processor	32
XBE-PxE.RD_PAR<3..0>	Parity of return data	4
XBE-PxE.CM_STATUS	Commreg status to processor	1
XBE-PxE.CM_STATUS_RDY	Commreg status valid	1
XBE-PxE.REQ_PEND	Pending requests active	1
XBE-PxE.ST_PEND	Pending Store requests active	1
PxO-XBO.RDY	Processor has a valid odd request	1
XBO-PxO.REQ_NEXT	Crossbar can accept req next clock	1
PxO-XBO.ADDR<28..3>	Processor request address	26
PxO-XBO.BD_SEL<3..0>	Processor board select	4
PxO-XBO.WR_DATA<31..0>	Processor write data	32
PxO-XBO.WR_PAR<3..0>	Processor write data parity	4
PxO-XBO.WR_ZONE<3..0>	Processor write zones	4
PxO-XBO.CYCLE<1..0>	Processor operation code	2
PxO-XBO.CTL_PAR<4..0>	Parity for Address and Control	5
XBO-PxO.RD_RDY	Return data ready for processor	1
XBO-PxO.RD_DATA<31..0>	Return data for processor	32
XBO-PxO.RD_PAR<3..0>	Parity of return data	4
XBO-PxO.REQ_PEND	Pending requests active	1
XBO-PxO.ST_PEND	Pending store requests active	1

Table 1-6: Zone mapping

Zone Bit	<0>	<1>	<2>	<3>
WR_DATA Byte	<31..24>	<23..16>	<15..8>	<7..0>

Table 1-7: Board Select

BD_SEL	Board
0	Memory Board 0
1	Memory Board 1
2	Memory Board 2
3	Memory Board 3
4	Memory Board 4
5	Memory Board 5
6	Memory Board 6
7	Memory Board 7
8	Commreg board

Table 1-8: Memory Operations

CYCLE	Mnemonic	Operation
00	NOP	No operation for memory
00	CS	Commreg status for commreg board
01	READ	Read
10	WRITE	Write
11	TAM	Test and Modify (TAS or TAC)

BD_SEL, ADDR, CYCLE, WR_ZONE, WR_DATA and WR_PAR define a memory request from a processor. BD_SEL, as given in table 1-7 , determines which board the request is to be made to. ADDR is the address that is to be written or read. If BD_SEL indicates the request is to a memory board, bits <6..3> will become the bank select as indicated in Table 1-1 and bits <28..10> will become the address field to the memory board. The remaining address bits are not used for memory accesses. If the request is for the commreg board, all bits are used.

Memory boards accept four types of requests. These request types are coded in the CYCLE field as given in Table 1-8 . In the case of a READ, the write data is not relevant, although it must, at all times, have good parity. For a write, the ZONE field indicates which of the four bytes in the data word to actually write. A code of all zero is interpreted as a NOP (no operation by the memory boards). The commreg board interpretes an all zero cycle field as a class of operation which return status information on a commreg lock bit without actually reading or writing any commreg data (see the commreg functional document for further information).

Table 1-6 shows which zone bits control which data byte. For a TAM operation, the processor will be returned the data at the requested address, then the data in WR_DATA under ZONE control is written into memory in an atomic operation.

WR_PAR is a bit of odd parity for each of the data bytes of WR_DATA. WR_PAR must be valid for WR_DATA on all clocks. WR_PAR is mapped to WR_DATA in the same way that WR_ZONE is.

When return data is valid for a processor, the crossbar will assert RD_RDY and place the return data on RD_DATA. Valid parity will always be maintained for RD_DATA. Read parity is mapped identically to write parity.

There is no handshake for the acceptance of return data. The processor must accept the return data on the clock RD_RDY is asserted. Since the processor will always accept return data, there are no overrun registers in the return data path. Return data flows without interruption from memory board through the crossbar to the processor.

The PEND signals indicate that the crossbar queues for a processor are not empty. ST_PEND is asserted if any stores are in the queue while REQ_PEND is asserted if any requests are in the queue, regardless of the type of the store.

1.1.3.3 Crossbar Memory Interface

The crossbar to memory board interface is similar to the crossbar to processor interface both in timing and general purpose of the signals. All signals are to be register to register with the exception of RD_DATA and RD_PAR. These two signals, in order to save a clock of latency, will be allowed to first go through a fast 9:1 multiplexor before being registered. The multiplexor and register must be within the same gate array.

ADDR, WR_DATA, WR_PAR, WR_ZONE and CYCLE are the values given to the crossbar by the processor at the time the original request was made.

A DONE signal is associated with each bank of a memory board, even and odd. When the crossbar sends a request to a bank, it marks that bank busy. It can send no more requests to that bank until that bank asserts its done bit. The DONE signal from the bank controllers on the memory board will be advanced so that full bank bandwidth can be utilized. Therefore the DONE implies that any request sent after DONE is asserted will arrive at that bank on the clock following the last clock of the bank's precharge time.

The crossbar internally keeps track of when return data is expected from a memory board. The RD_RDY signal is therefore unnecessary. It is maintained for debugging purposes and to provide a measure of functional error checking. No return handshake is required between the crossbar and memory board. A return ready (RD_RDY) signal *is* generated for the processor itself, however (and is necessary).

1.1.3.4 Crossbar to Communications Register Interface

Table 1-9: Crossbar to Commreg I/O

Signal	Description	Width
XBE-CRE.RDY	Crossbar has even board request	1
XBE-CRE.ADDR<28..3>	Request address	26
XBE-CRE.WR_ZONE<3..0>	Request write zone	2
XBE-CRE.WR_DATA<31..0>	Request write data	32
XBE-CRE.WR_PAR<3..0>	Request write data parity	4
XBE-CRE.CYCLE<1..0>	Request operation type	2
XBE-CRE.CTL_PAR<4..0>	Request parity for control	5
CRE-XBE.RD_DATA<31..0>	Return data	32
CRE-XBE.RD_PAR<3..0>	Return data parity	4
CRE-XBE.RD_RDY	Return data available	1
CRE-XBE.Px_CM_STATUS	Commreg op lock bit result	1
CRE-XBE.Px_CM_STATUS_RDY	Commreg op lock bit result rdy	1
XBO-CRO.RDY	Crossbar has even board request	1
XBO-CRO.ADDR<28..3>	Request address	26
XBO-CRO.WR_DATA<31..0>	Request write data	32
XBO-CRO.WR_PAR<3..0>	Request write data parity	4
XBO-CRO.CYCLE<1..0>	Request operation type	2
XBO-CRO.CTL_PAR<4..0>	Request parity for control	5
CRO-XBO.RD_DATA<31..0>	Return data	32
CRO-XBO.RD_PAR<3..0>	Return data parity	4
CRO-XBO.RD_RDY	Return data available	1

The communications register (commreg) board is a special purpose board. Its interface is similar but not identical to a memory board's. The commreg board contains two basic functions- the communication registers themselves and I/O space.

The commregs are a small set of memory supporting atomic test-and-modify operations on each of its elements. Of the full 26 bit address sent to the commreg board, the commreg's themselves only require 16 bits for addressing. The remaining address bits encode the processor ID (PID) of the processor that issued the request and the several cycle extension bits that code the full range of commreg operations.

The processors are constrained to always make longword requests to the commreg board (commreg and I/O space). Furthermore, the request can only be made if the crossbar queue for the processor is empty (REQ_PEND not asserted). Since the arbitration in the crossbar is functionally identically between even and odd sides and is performed at the memory/commreg board level, these two conditions will ensure that both words of all commreg requests reach the commreg board together. Without these restrictions words of a longword commreg request might arrive at different times with other processor's requests intervening.

Some of the commreg operations require the return of lock bit status without any return data. To accommodate this data, there is a CM_STATUS, CM_STATUS_RDY pair of signals from the commreg board to each processor. These signals are simply repeated on the crossbar board in free clocking registers and do not use any part of the normal return data path.

All commreg board operations, be they to commreg or I/O, require fewer clocks to process than memory board operations since the commreg board uses ECL static rather than dynamic RAMs. The general crossbar return data scheme assumes that all return data comes back a uniform amount of time later. This assumption greatly simplifies return control. Commreg returns, for which the assumption is not true, are treated as a special case. As will be described in the section on return data, commreg data is inserted in the return stream as close as possible to its ideal, fast return time without destroying the FIFO nature of returns.

Commreg return latency is constant. Since it is constant, the return controller in the crossbar can predict when commreg return data is expected as in the case of memory board returns. Thus, the RD_RDY signal from commreg board to crossbar is also unnecessary but is maintained for debug and error checking.

Unlike the done mechanism used by the memory boards, the commreg/crossbar interface uses only a RDY signal to inform the commreg board that there is a request for it. No REQ_NEXT or DONE is necessary since the commreg can indefinitely accept requests at the rate of one per clock.

As a design contingency, the ARB gate arrays will be able to accept a REQ_NEXT handshake for sending requests to the commreg board to allow for a commreg design that can not accept requests at the rate of one per clock. Adding REQ_NEXT for the commreg port should not have much of an impact on the ARB gate array design. Functionaly, if a REQ_NEXT is required for sending requests to the commreg then returns from the commreg board are not easily predictable. The RTNCTL must be able to accept commreg return data based on the RD_RDY from the commreg board (instead of internally predicting when the data is to return). The RTNCTL gate array is sufficiently light on pins and cells to allow the inclusion of logic for both the fixed return time and non-fixed return time commreg options. The actual logic function used would then have to be scan selected.

1.1.3.5 Arbitration

Each clock, on its nine even and nine odd processor ports, the crossbar is able to accept nine requests. As long as no two requests are to the same memory board or no request is to an unavailable (busy) memory bank, the crossbar can continue to accept requests without interruption. When multiple requests are to the same memory board, the crossbar must then choose one processor to receive access to the requested board and push the other processors' requests to following clocks. If a processor wants access to a busy bank, it will be held up until the bank indicates that it is done with its current request. Therefore arbitration is performed across all processors requesting idle memory banks on the same memory board.

When a processor's request is held up, the REQ_NEXT signal will be de-asserted on the following clock and continue to be de-asserted until the request is able to proceed. Requests made by the processor while REQ_NEXT for the previous clock was asserted will, of course, be accepted. This necessitates several levels of overrun registers within the crossbar to accommodate these requests issued after a request is blocked but before REQ_NEXT informs the processor of the block.

The arbitration process for the even and odd sides of memory is completely independent of each other. It is possible for a processor to have a request blocked on one side and that side's REQ_NEXT de-asserted while the other side is still accepting requests. It is also possible for one processor to request the even side of a memory board while a different processor requests the odd side of the same memory board without either processor blocking.

When multiple processors request the same board on the same clock, the crossbar must arbitrate between the two requests. Arbitration for contended memory boards is round robin- each processor receives its turn as highest priority. *However*, once a processor wins access to a memory

board, it is allowed to keep access to this board as long as it continues making requests to this board at the rate of one per clock. A processor is limited to sixteen sequential requests before automatically losing arbitration to prevent a processor from locking up a memory board (this number is scan programmable and will probably be increased for single board systems).

The above arbitration scheme has proven superior in simulation to a strict clock by clock round robin. This "burst" scheme allows a processor doing a vector operation (or an I/O transfer) to get quickly through a board. In a case where all processors are doing sequential longword accesses, this scheme allows the processors to spread out so that only one processor requires a particular memory board at any one time instead of all processors hammering away at the same board. The burst length will be scan settable (from one which becomes pure round robin, to one hundred twenty eight) in the crossbar.

1.1.3.6 Send Path Contention

Two situations can interfere with the flow of requests from the processors to the memory and commreg boards. Since there is only a single port into an even or odd side of memory, if multiple processors request the same port on the same clock, only one processor can be granted access while the other processors must wait. Additionally, if a processor is requesting a busy bank (or an unready commreg board), that request will also be blocked.

When a processor has a request held, REQ_NEXT will be de-asserted on the following clock. It will be re-asserted when the request as soon as the request at the head of the processor's queue is able to proceed. In the meantime, the crossbar must accept any requests made while REQ_NEXT was being generated and transmitted to the processor. Each processor port into the crossbar therefore has a register for the current request and overrun registers for any requests that are made before the processor sees REQ_NEXT. If the processor made requests immediately after issuing the request that blocked, it is possible that there will be requests still in the crossbar queue when REQ_NEXT is reasserted. There will, of course, be slots available in the queues to accept all valid requests at all times.

Between the crossbar and memory, the internal busy maintained by the arbitration logic and the DONE from the memory board govern whether a request can be made. If the required bank is not busy, the request can be made and will be accepted at the bank state registers. That bank can not then be requested until it signals that it has finished (is DONE) with the previous request. Therefore, aside from the state at the bank level necessary to process a request, there is no staging or overrun for requests to a memory board. Requests can only be made if they can be immediately serviced.

From performance simulations of the proposed memory system, it was found that mean return time for read data was about 2.5 clocks greater than the ideal return time. Median return time was generally the ideal return time. The ideal return time is shortest return data latency possible; it occurs when the read request is not blocked or put in an overrun register in the crossbar and the request is to an idle bank. Ideal return time is approximately 19 clocks from processor memory interface making its request to the interface receiving its data.

The upper bound on the amount of time a request may wait in the crossbar before being sent to a memory bank is:

$$n\_burst * (n\_proc - 1) * t\_rmw + (q\_depth - 1) * t\_rmw$$

where *n_burst* is the maximum burst allowed by arbitration, *n_proc* is the number of processors in the system, *q_depth* is the amount of staging in the crossbar for unserviced requests, and *t_rmw* is the cycle time for a read-modify-write operation such as a byte store or a test-and-set. For expected numbers of, respectively, 16, 9, 4, and 21; maximum wait time is 2751 clocks or about 44 microseconds. Such a case only arises when every processor makes a request at the same time and all requests are read-modify-write operations to the same bank.

Maximum return data latency is therefore:

n_burst * (n_proc - 1) * t_rmw + (q_depth - 1) * t_rmw + t_read + t_stage

where *t_read* is the read access time of the RAMs and *t_stage* is the staging time to get the request to the bank from the crossbar and from the bank to the processor. In general, *t_read* is about equal to *t_stage*. The last two addends are negligible. With the earlier expected values, maximum read latency is also on the order of 44 microseconds.

1.1.3.7 Return Data Path

Since requests are only made to unbusy banks, and all banks have the same cycle time, at most one piece of return data per even/odd side per processor may be ready on any one clock with the exception of commreg return data which will be dealt with later. Therefore, the multiplexor that selects return data for a processor from the eight memory boards and the commreg board need only select a single source; there is no contention for this multiplexor.

To control the return data multiplexor and to generate the RD_RDY signal to the processor, the crossbar maintains queues of the memory board ids that read requests were made to. These queues are simply delay lines of linked registers with programmable taps to allow for variations in DRAM access times. If at time *n* a processor's read request is allowed to progress to the memory boards, that request will return at time *n* + *t_read_latency*. The memory board id queues simply stage the board id that the request was made to so that at time *n* + *t_read_latency* the multiplexor is selecting the proper board. In addition to the board id, the queues also register whether return data is, in fact, expected at time *n* + *t_read_latency*. If it is, a RD_RDY to the processor is generated at the proper time. These queues are maintained for each processor.

Since *t_read_latency* is lower for commreg requests (but is also a constant for all types of commreg requests) than it is for memory requests, commreg requests will be available earlier than memory read requests. It is therefore possible for two pieces of data to be ready on the same clock, one from one of the memory boards and one from the commreg board which introduces contention for the return multiplexor.

The contention is resolved by queueing commreg return data in the return data gate arrays. The commreg data is only allowed to proceed if no memory data is expected on the current clock and all requests made before the commreg request have already returned their data. The processor memory id queues are used to determine both conditions. The second condition is required to ensure that the processor receives its data in the same order as it requested it.

Commreg data can therefore return, ideally, as quickly as the minimum latency of the commreg board will allow. However, earlier memory load requests may prohibit the commreg data from returning at its earliest possible time. The most a commreg request can be deferred is the difference between memory board latency and the commreg latency. Therefore the commreg data staging queues in the return path need only be as deep as this difference. This limit holds because a processor, at any one time, can only issue a single request to all the memory boards and the commreg board. So, on the clock that the commreg request was issued, no request could have gone to memory and, therefore, *t_read_latency* clocks from the commreg request no memory data can be returning. This provides a guaranteed hole for the commreg data to use; this hole being *t_read_latency* clocks from the issuance of the commreg request.

Since the processor will always accept return data, the above scheme allows all memory board return data to pass directly through the crossbar without any possibility of being held. Commreg data may be held, however there is a small upper bound on the amount of time this data may be held. Commreg return data will take no more time to return than memory return data.

1.1.3.8 Valid Physical Address Checking

On the C2, valid physical addresses were checked by cycling each access against the Physical Configuration Map (PCM) on the cpu utilities board. For software, a shadow PCM may still be maintained but actual detection of invalid addresses will be performed in the crossbar, the memory boards and the commreg board.

The crossbar will detect accesses to non-existent memory boards. The memory boards will detect accesses to non-existent RAM. Since the memory is only a single row per bank and only 1M and 4M RAMs are to be supported, it is only necessary to check two address bits and only when 1M RAMs are installed. When 4M RAMs are used, there is no unimplemented memory within a memory board. The commreg board will detect illegal accesses to I/O space.

When an illegal address is detected, a hard error from the appropriate memory board or the crossbar will be generated.

1.1.4 Parity Checking

Table 1-10: CTL PAR mapping

PAR Bit	Control Bits Covered	# Bits in group
0	ADDR<28..22>	7
1	ADDR<21..15>	7
2	ADDR<14..8>	7
3	ADDR<7..3>::CYCLE<1..0>	7
4	ZONE<3..0>	4

Parity checking is only performed at the memory boards and at the processors. No parity checking is done within the crossbar itself. Both data and control is covered by parity. Data parity bits are RD_PAR and WR_PAR previously described. ADDR, ZONE, and CYCLE are covered by the CTL_PAR bits. The mapping between CTL_PAR and the actual bits covered is given in Table 1-10 . All parity is odd parity.

While the crossbar does not perform parity checks, it will maintain enough state to determine where the data with the bad parity originated and what that data was when it was in the crossbar should a parity error be detected at the processor or memory board. This will greatly aid in isolating the cause of the failure. The state required to determine this information is the processor selects for the memory boards on the send path, as well as the send information itself. On the return path, the memory board select and return data must be saved. All of this data must be available for a number of clocks after the data was actually sent to the memory board or processor. The number of clocks necessary will be determined by the latency between the parity error detection and the resulting clock stop on the crossbar board.

1.1.5 Cycle and Access Times

Table 1-11: Approximate Cycle and Access Times, 80ns RAMs

Operation	Cycle Time	Access Time (at crossbar)	Access Time (at processor)
NOP	15	–	–
Read	15	13	19
Read w/ECC error	15	13	19
Full Write	15	–	–
Partial Write	21	–	–
Partial Write w/ECC error	42	–	–
TAM	21	13	19
TAM w/ECC error	42	13	19

Table 1-11 summarizes the cycle and access times for the memory system. A more thorough discussion is found in the memory system detailed specification document. The table assumes DRAMs with an 80 ns access time which are the slowest RAMs suitable given a design cycle time of 12 ns. The first access time column gives the number of clocks from the *crossbar* issuing a read request to a memory board to the crossbar receiving that data from the memory board. The second column gives the times with respect to the memory interface at the processor port to the crossbar. The times of the second column are equal to the first column times plus the number of pipeline stages through the crossbar. Times given in the table for ECC errors are for single bit, correctable errors.

1.1.6 Staging in the Crossbar

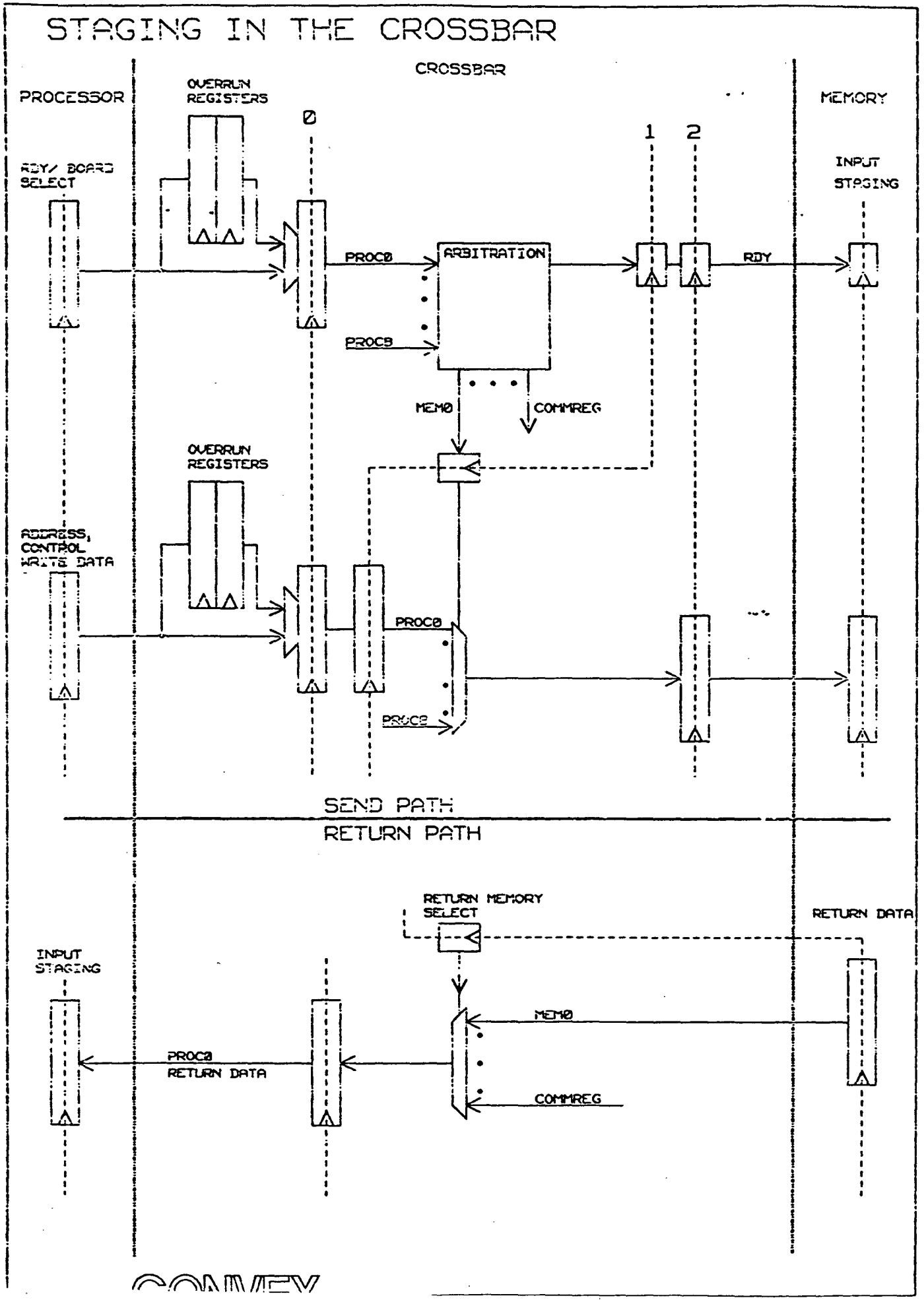
The physical separation of the crossbar from the processor and memory boards is such that an entire clock is required for propagating a signal between the crossbar and any other board. The process of arbitrating requests from the various processors also consumes clocks. Figure 1-3 shows the registers in the processor-crossbar-memory (send) path and the memory-crossbar-processor (return) path. In this figure, registers linked by thin, dashed lines are equivalent in time. The diagram has been somewhat simplified but is functionally correct.

From data valid at the processor's connector to data capture in the memory board's input staging register, four clocks elapse. Two of these clocks are used to propagate data to and from the crossbar itself. The remaining two clocks are used to arbitrate the requests and select data for the memory port. On the return path, only the two clocks necessary to travel to and from the crossbar are required. The selection of return data for the processor port is performed on the same clock that the data is registered from the memory board.

Following a single read request from processor to memory and back to processor, it can be seen that six clocks are required just to transit the crossbar. On the first clock, the processor places its request state on the bus to the crossbar, driven from the registers on the upper left edge of Figure 1-3. If there has been no contention recently at this processor port, the request state is registered in register set *0* in the crossbar. The upper register *0* is in the ARB gate array. This gate array receives the RDY, BD_SEL and bank select bits of ADDR. Remaining bits are captured in the lower register which is physically located in the eleven SEND_XBAR gate arrays.

On the second clock, the request will be arbitrated. If it wins arbitration, the processor that made the request will be selected for the memory board it was requesting; the processor's id will be placed in the register *1* below the arbitration block. In the SEND_XBAR, the request state is moved forward one register. On the third clock, the registered select selects the data for the level *2* registers. On the fourth clock, data propagates from the register *2* to the input staging registers on the memory board.

STAGING IN THE CROSSBAR



Approximately thirteen clocks after the read request was sent to the memory board, the read data for that request will be in the return data register and will propagate from the memory board to the crossbar. Before being registered in the crossbar, this data will be routed to the appropriate processor's register in the RTN_XBAR gate array. The selects for this return routing are available well in advance of the return data since the crossbar models the return time from all the memory board; it does not wait for a read ready signal from a memory board. Finally, the return data propagates from the RTN_XBAR register to the processor's input staging register.