C6-1476.16/33

N.I.

C6-1476.16/33
Copy 12

STUDY OF SPACEBORNE MULTIPROCESSING

SECOND QUARTERLY REPORT - PHASE II

Prepared under Contract No. NAS 12-108 by

Autonetics Division of North American Rockwell Corporation

Electronics Research Center
National Aeronautics and Space Administration

C6-1476.16/33

STUDY OF SPACEBORNE MULTIPROCESSING

SECOND QUARTERLY REPORT - PHASE II

30 Oct 1967

L.J. Koczela

Principal Investigator

Approved By:

G.B. Way

G.B. Way
Chief Engineer
Data Systems Division

Prepared under Contract No. NAS 12-108 by

Autonetics Division of North American Rockwell Corporation
3370 Miraloma Avenue, Anaheim, California 92803

Electronics Research Center
National Aeronautics and Space Administration

FOREWORD

This second quarterly report describes the work accomplished during the second quarter of Phase II under NASA contract NAS 12-108, Spaceborne Multiprocessor Study. It was performed by Autonetics, a division of the Aerospace and Systems Group of the North American Rockwell Corporation. The work was administered under the direction of the National Aeronautics and Space Administration, Electronics Research Center, Computer Research Laboratories, Cambridge, Massachusetts; the NASA project manager is Mr. G.Y. Wang.

The contract participants during this quarter and their primary responsibilities are listed below:

L.J. Koczela	-	Parallelism, Input/Output, Communication Operation
P. Bogue	-	Architecture, Communication Operation
G.J. Burnett	-	Macro Instructions, Processor Design

CONTENTS

ILLUSTRATIONS

TABLES

1. INTRODUCTION

This report presents the results of the activity during the second quarter of the Phase II portion of the Spaceborne Multiprocessor Study. The purpose of the Phase II effort is to perform a detailed investigation of the distributed processor computer organization. In particular, the following specific tasks are to be covered during this phase: detailed system analysis; organization logic design; failure detection, isolation, and reconfiguration; and software analysis.

The previous quarterly report included a qualitative and quantitative description of the computer requirements, a discussion of the semiconductor technology extrapolated ten years out, a discussion of parallelism within computations and methods of analyzing the computations for parallelism, and the development of computer organizations; in particular the distributed processor computer concept, was presented.

Figure 1-1 contains a block diagram of the organizational concept. The organization is seen to consist of a number of identical cells interconnected in a particular manner. Each cell consists of a general purpose processor section and a small amount of memory (512 16-bit words) on a single MOS wafer. The cells are divided into groups (4 groups of 20 cells each are considered for the spaceborne application) and these groups are connected by an intergroup bus for communication. Within each group the cells communicate with each other by an intercell bus and by neighbor communication lines. Each group will have one cell designated as a controller cell; the remaining cells can be operated independently or dependently of the controller cell. This organization is thereby capable of simultaneously taking advantage of applied (global control) parallelism and natural (local control) parallelism within computations. It offers extremely high reliability for space missions by having many levels of graceful degradation (tolerant of many internal failures before resulting in computer system failure). It also results in low power consumption due to the ability to turn cells on and off to closely match varying computational requirements and also provides a system capable of being applied to a wide variety of missions due to the flexibility of the number of cells and groups comprising the organization.

This report presents the results of the applied and natural parallelism investigations. Neighbor-to-neighbor communications were included in the organization and a discussion of this topic is contained in this report. The input/output scheme was investigated in depth and a preliminary description of the processor section prepared. The architecture of this organization is presented herein with a description of the general operation of the computer system. In addition, a description of the communication bus operation is included in this report.

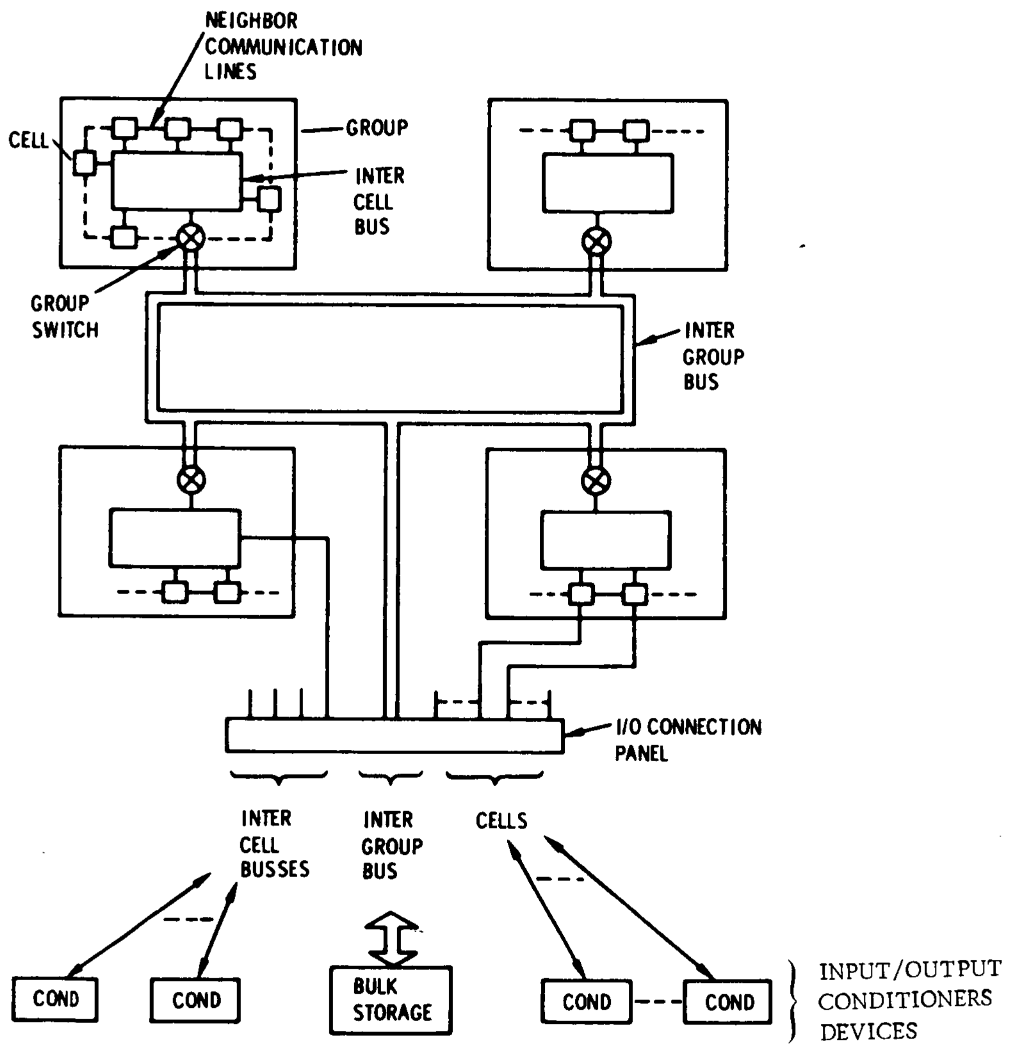


Figure 1-1. Distributed Processor Organization

2. PARALLELISM STUDIES

Section 4 of the last quarterly report contained a discussion of parallelism in general and methods for analyzing computations to determine the amount of parallelism within them. At that time the computations for the manned Mars lander mission were being analyzed for parallelism. The results of this analysis are included in this report. Each of the computational tasks as defined in the requirements in the Phase I study were investigated and the results are summarized in Tables 2-1 and 2-2.

Figures 2-1 and 2-2 contain the applied parallelism speed curve; they show the computation reduction ratio vs the degree of applied parallelism available in the computation system. It should be noted that Figure 2-2 replaces Figure 4-16 of the first quarterly report since the latter was a curve of preliminary results. The 100 percent utilization curve in the figures is the 1-to-1 curve, i.e., for a degree of parallelism of 2 the computation reduction ratio would be 2, for 5 it would be 5, etc. The actual curve is seen to deviate slowly from the 1-to-1 curve at first and then reaches an asymptotic reduction ratio value of 13.66 for higher degrees of parallelism. The knee of the curve occurs at approximately a degree of 15; beyond this degree the curve deviates sharply from the 1-to-1 curve.

Figures 2-3 and 2-4 contain the computation reduction ratio and storage required per cell (assuming one degree of parallelism results in one cell), respectively, vs the degree of natural parallelism available in the computation system. It should be recalled that natural parallelism includes applied parallelism by definition. The vertical scale in Figure 2-3 is the same as in Figure 2-1; however, note that the horizontal scale is considerably larger. The computation reduction ratio curve does not

Table 2-1. Applied Parallelism Results

Degree of Applied Parallelism	Computation Reduction Ratio
1331	13.66
800	13.66
300	13.65
100	13.47
50	12.58
15	8.75
5	3.98
2	1.895
1	1

Table 2-2. Natural Parallelism Results

Degree of Parallelism	Computation Reduction Ratio	Storage (Words/Cell)
1342	42.4	
300	42.4	200
100	42.4	350
50	28.2	575
15	12.15	1,800
5	4.6	5,250
2	1.96	12,700
1	1	24,633

have as sharp a knee as in the applied parallelism case. However, it appears that somewhere in the range of 40 to 60 in degree of parallelism the curve starts to deviate rapidly from the 1-to-1 curve. The storage curve is drawn on log-log paper to bring out the deviations from the 1-to-1 curve. It can be seen that the curve begins to deviate rapidly from the 1-to-1 curve in the vicinity of a degree of parallelism of 80 to 150.

The above curves give an indication of the efficiency or utilization of parallelism. They may also be used in determining the speed-storage characteristics of the cells. It should be recalled from the technology section of the first quarterly report that a storage capability of approximately 512 words per cell was considered to be achievable. If this storage is available per cell, then, referring to Figure 2-4, one can see that approximately 58 cells are needed (assuming a degree of parallelism equals one cell). Translating this into speed requirements one can see from Figure 2-3 that a computation reduction ratio of approximately 31 results with 58 cells. Since the speed requirement for a single computer is approximately 1,450,000 short operations/sec (recall that the parallelism investigations were carried out for the Mars orbital phase which has the maximum speed and storage requirement), the speed requirement per cell is therefore approximately 25,000 short operations (ADD, SUB, etc.) per second. Since these requirements did not include overhead functions (such as the executives) one may estimate the number of cells required at approximately 80 with a storage capability of 512 words each and requiring a speed capability of 25,000 operations per second. One can now see that the cells will most likely be storage restricted rather than speed restricted since the cells should be capable of more than 25,000 short operations per second.

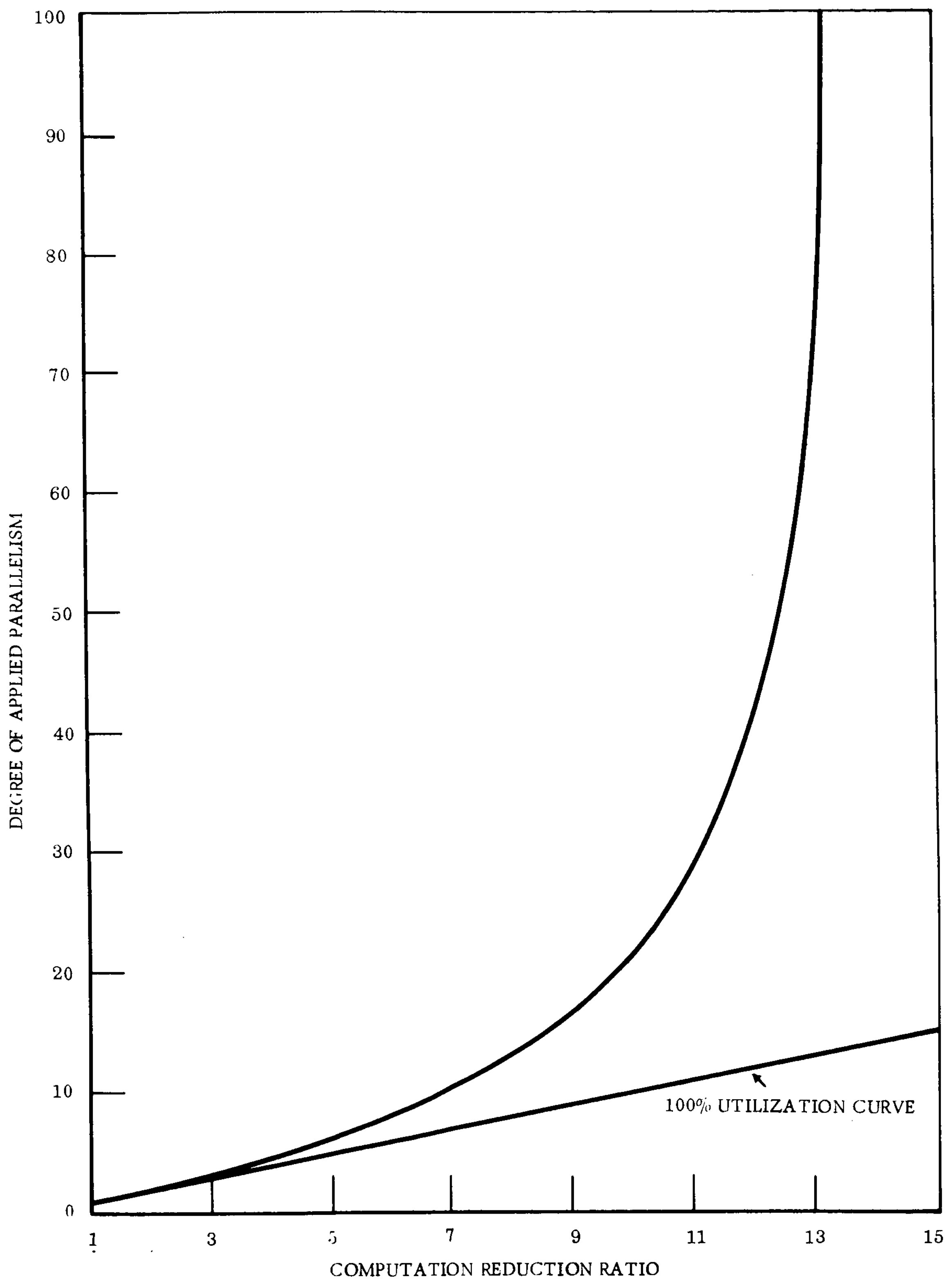


Figure 2-1. Applied Parallelism Speed Curve

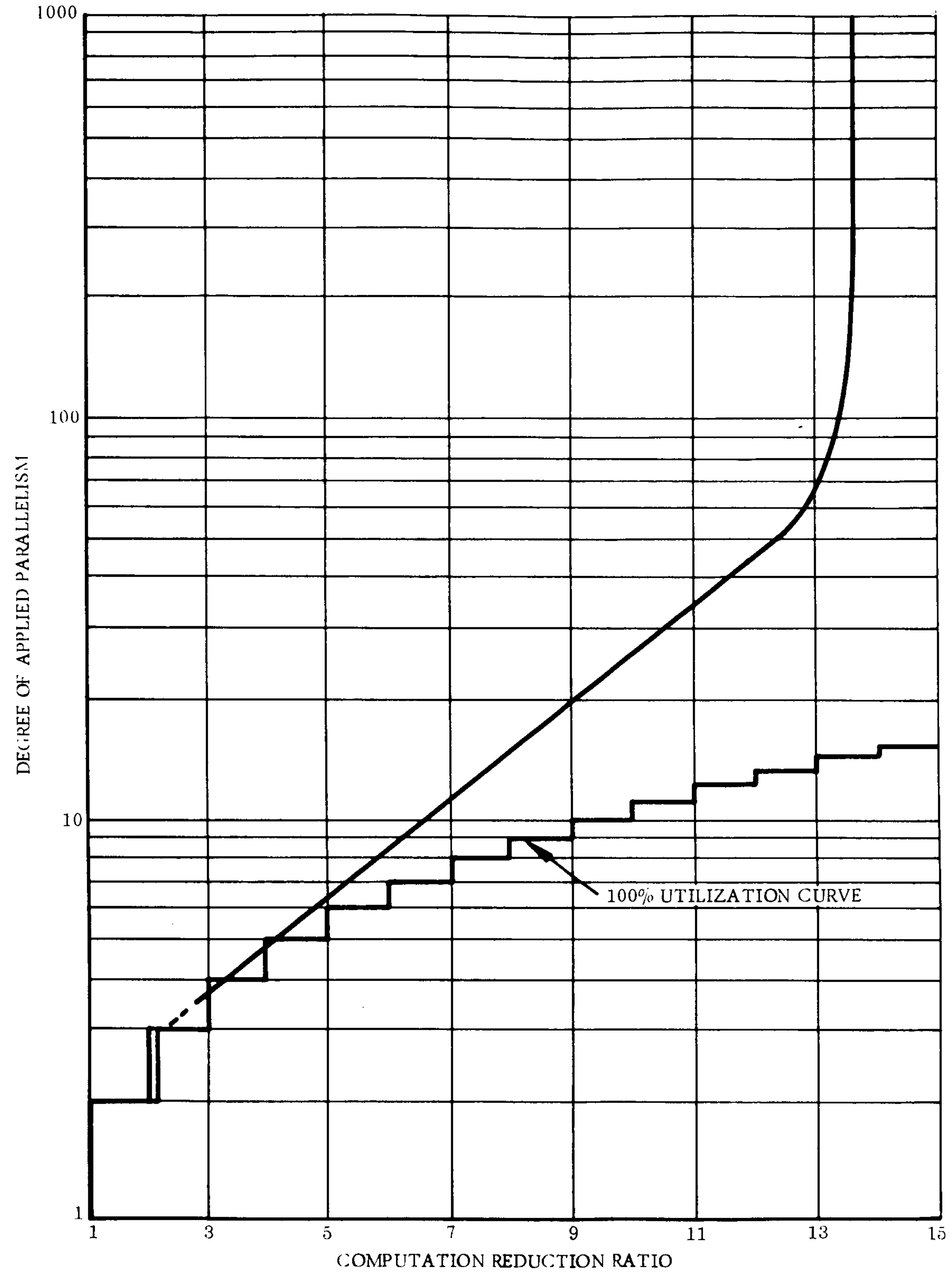


Figure 2-2. Applied Parallelism Speed Curve

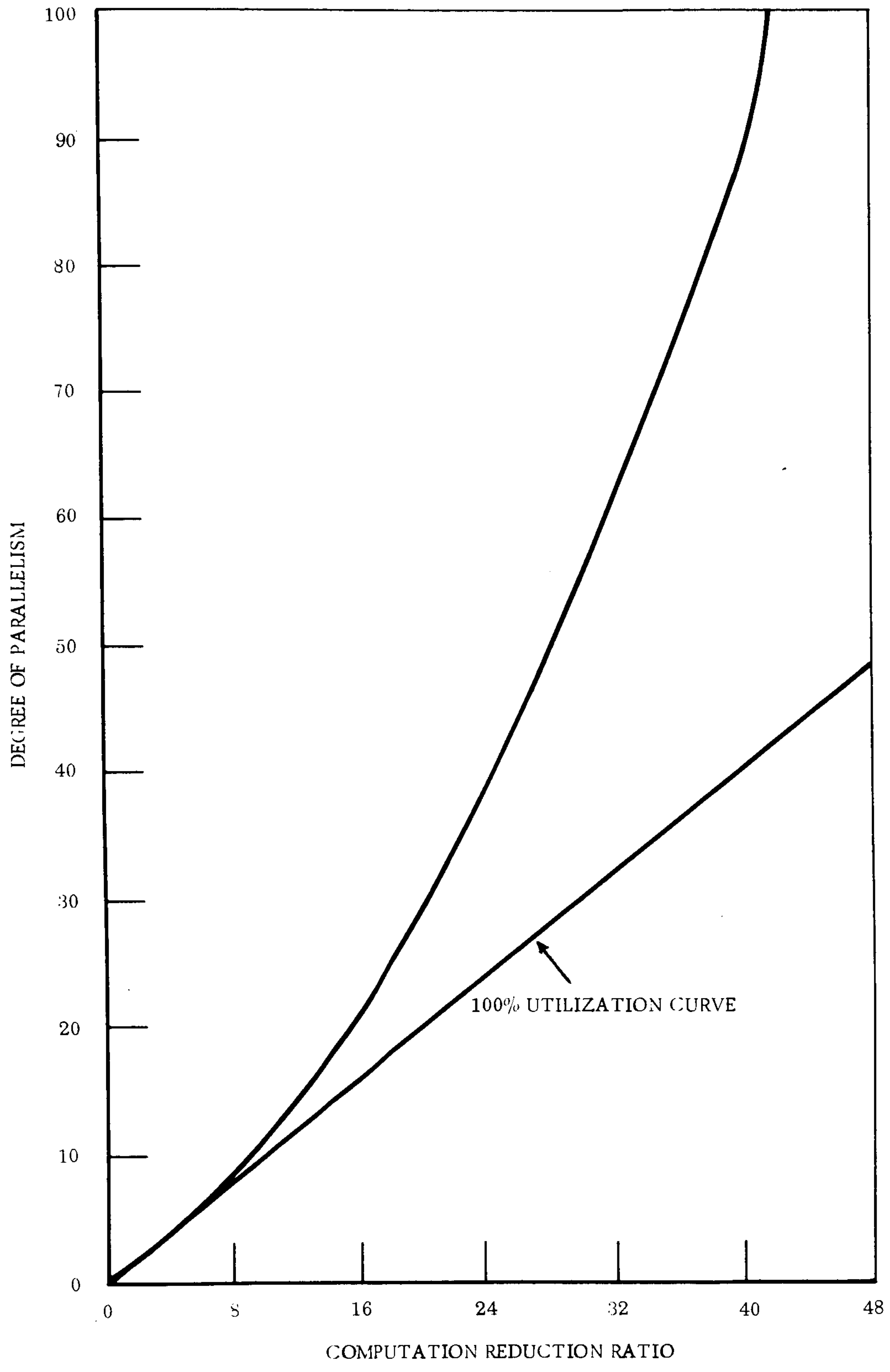


Figure 2-3. Natural Parallelism Speed Curve

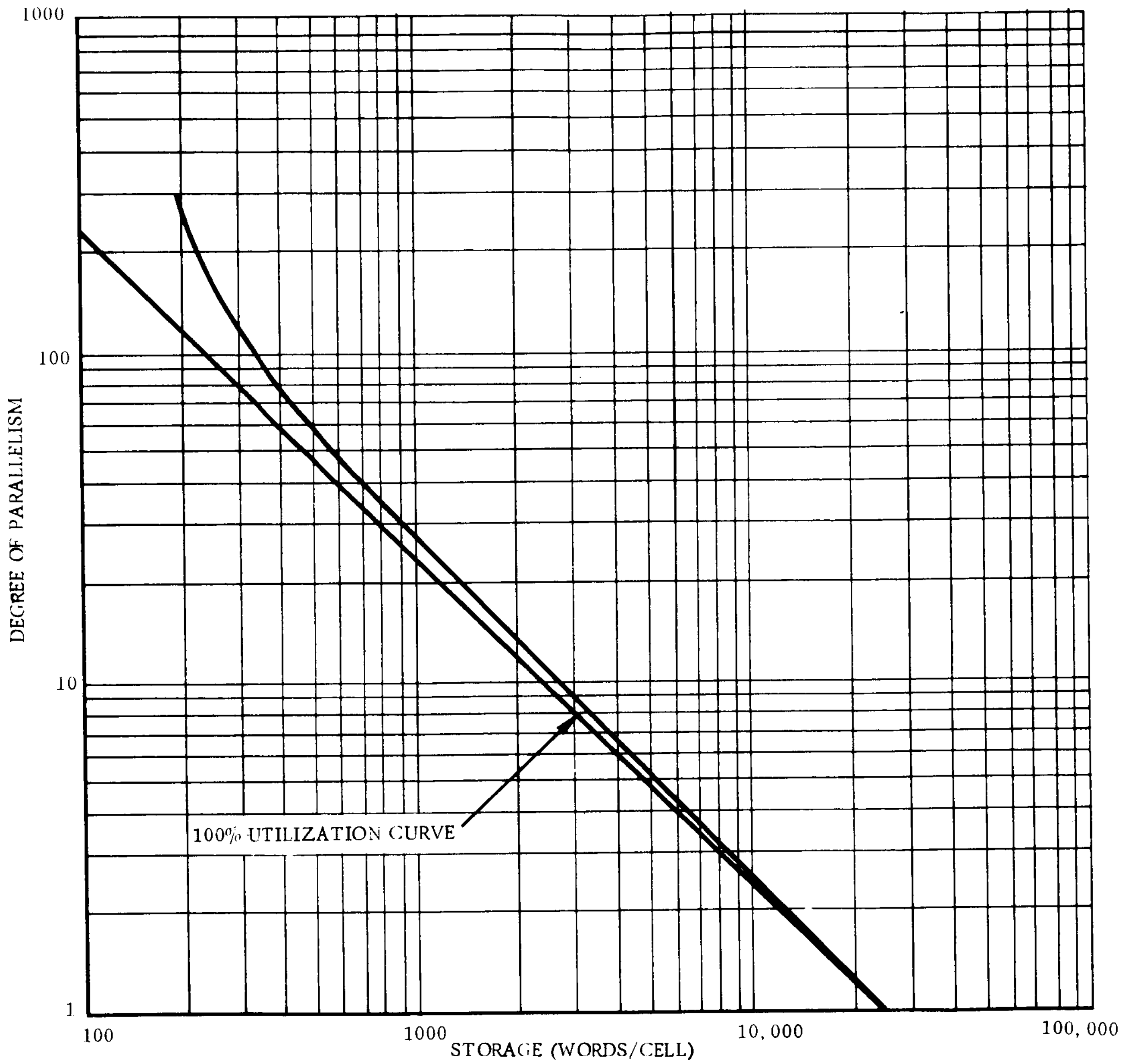


Figure 2-4. Natural Parallelism Storage Curve

3. NEIGHBOR TO NEIGHBOR COMMUNICATIONS

3.1 NEIGHBOR COMMUNICATIONS FEATURES

This section will discuss the reasons for including neighbor communications and some considerations as to its mechanization. Neightbor-to-neighbor communication is a means for one cell to communicate with a restricted number of neighboring cells. These communications paths are separate from the intercell bus. Because the bus is always needed for global instructions (applied parallelism), the question arises, should special communication paths be provided to neighboring cells, or should all communication go via the intercell bus?

The advantages of neighbor-to-neighbor communication come from the problems of using the bus and the fact that certain computations may be placed with a geometric relationship (matrix and vector manipulations, etc.) that may efficiently utilize neighbor communications. The intercell bus requires control as to which cell gets to use the bus, how long the cell may use the bus, and how control is to be passed from one cell to another. A controller cell must have means of controlling transmission priorities. Even if some of this control can be done by hardware, the fact remains that there will always be an overhead in software and storage. To address any word in a group requires 14 bits; thus to send a 16-bit data word requires 14 bits of storage to hold the destination address. Thus for a word of data, a word of address is required. Adding words of program to control the transmission will require additional words of storage.

The time delay in transmitting via the intercell bus will depend upon the bus design and how long it takes to establish control. If the bus is 8 data bits wide, for example, about 6 clock times are required to transmit a single word. When many words are transmitted in a single message, the clock times per word will approach 2 (one clock time per 8 bits). A detailed discussion of how communication is carried out on the bus may be found in Section 6.

The time delay in transmitting is seen to be a problem, especially when few words are transmitted. A more serious problem is the delay in obtaining control of the intercell bus. The time delay between the time a cell needs the intercell bus and the time a cell actually gets control of the bus is called the request delay. The longest delay will occur when all the other cells have transmissions to make and the requesting cell must wait its turn. The shortest time can be almost zero. Somehwere between these two times will be the average request time delay. This delay is variable, unknown, and could be very large. For any program where the timing is critical, the unknown times could make the programming difficult. In many cases where data has to be passed to neighboring cells the timing is critical, such as in the navigation and guidance routines. It should be noted that the more applied parallelism that is being utilized, the longer the request delay. Since this may mean that more cells are involved in the neighbor communications; if they all had to be serviced over the intercell bus, the average request delay can be very large.

It is of course possible to do without neighbor communications providing one does not run into a time problem and cannot complete the computations in the allotted time. However, as noted above, it would prove very inefficient and difficult to use the intercell bus where small amounts of data have to be passed between neighbors as in the use of applied parallelism.

To improve the system, neighbor-to-neighbor communication is needed. The system implemented must provide the following advantages:

1. Low overhead (no more than one instruction to execute a data transfer)
2. High reliability
3. Known times to implement a transfer
4. Adaptable to reconfiguration

The disadvantages to including neighbor communications are two: (1) additional connections are required to each cell, and (2) reconfiguration is more difficult. The number of connections required per cell increase by four; while this is not a large increase, it does of course provide more failure points. The most serious disadvantage appears to be that of reconfiguration since now the organization is spatially oriented. One approach to this problem is to require the programs to be set up using small independent sets of cells so that one may pack the sets of programs around a number of failed cells in a group. This problem will require further investigation in later phases of the study.

While a firm answer as to whether or not neighbor communications are required cannot be given at this time, the advantages in including it appear to outweigh the disadvantages that may be encountered. In addition, inclusion of the neighbor communications provides an organization with increased capabilities (notably speed) so that it may be applicable to a wider variety of applications (for example a high speed video data reduction problem).

3.2 IMPLEMENTING NEIGHBOR-TO-NEIGHBOR COMMUNICATION

It is seen above that any method proposed must have advantages over and above the intercell communication bus method. A method is proposed here that has these advantages. The functional operation is described here. One cell will pass a word of data to a neighbor cell. The sending cell, called here CO, will be executing a program PO. The receiving cell (C1) will be executing a program (P1) that requires the word of data contained in CO.

The program PO will place the word in one of its accumulator registers and set a flag associated with that register. This flag will indicate to whom the data is to be sent. Thus the flag can be set to 5 states: N, S, E, W, or X. The X setting is the normal state of the flag and indicates that the register contents are for the use of cell O only. Unless an instruction is executed to change the flag state, the registers can only be used by cell O; the flag will always be X. If the flag is set by an instruction to N, the North cell is to receive the register contents. By setting the flag to S, E, or W, the South, East, or West neighbor can receive the data.

Subsequently, the program in cell 1 needs the data from cell 0. Program P1 will execute an instruction with the following format.

OP CODE	R	C

OP CODE - Is the operation to be performed, and is any of the usual register-to-register operations (add, subtract, load, etc.)

R - Is the accumulator register into which the results will be loaded. This accumulator may also furnish an operand for some instructions.

C - Is the relative location of the neighbor cell (N, S, E, W) which will have a register with the proper flag.

The execution of this instruction will cause cell C1 to request from C0 a word of data. If C0 has a register with the proper flag, C0 will send C1 the data word. Each cell contains a buffer register (serial line used between cells); thus the transfer will not destroy any accumulator contents. After the transfer is complete, cell C1 will perform the operation specified by the OP Code, using the transferred word as one of the operands.

After a data word has been transferred the register flag in CO is reset to X. By executing an instruction, the cell can test its flags. In this way, a cell can verify that the neighbor cell called for and was sent the data. This flag test may provide a means of detecting a malfunctioning cell.

Several special cases are outlined below. The descriptions below may be changed as the system is studied further. Actual programming of some sample problems may show ways of improving the system.

If two accumulator registers are set to the same flag value, the lower numbered accumulator will be transferred first. Only one datum will be transferred at any one time for any single instruction execution.

CO may attempt to alter the accumulator contents before a neighbor cell has requested and received its data word from cell CO. The control circuits in CO can be set (via an instruction) to: (1) wait until C1 receives the data, (2) interrupt to some special error routine, or (3) ignore the flag and use the accumulator anyway. Which options are implemented will depend upon further study.

The hardware needed to implement this neighbor-to-neighbor communication scheme consists of a buffer register and control circuitry. Figure 3-1 shows a block diagram of the hardware. Upon execution of an instruction flagging an accumulator the hardware will set the flag flip-flops associated with the accumulator to the proper state. The control circuitry will be set to a state to expect a request.

Upon execution of an instruction requesting a word from a neighbor, the control in cell C1 will select the proper line and send a request to that cell. If cell CO does not have any flag set for this neighbor, the request will be rejected. If cell CO has

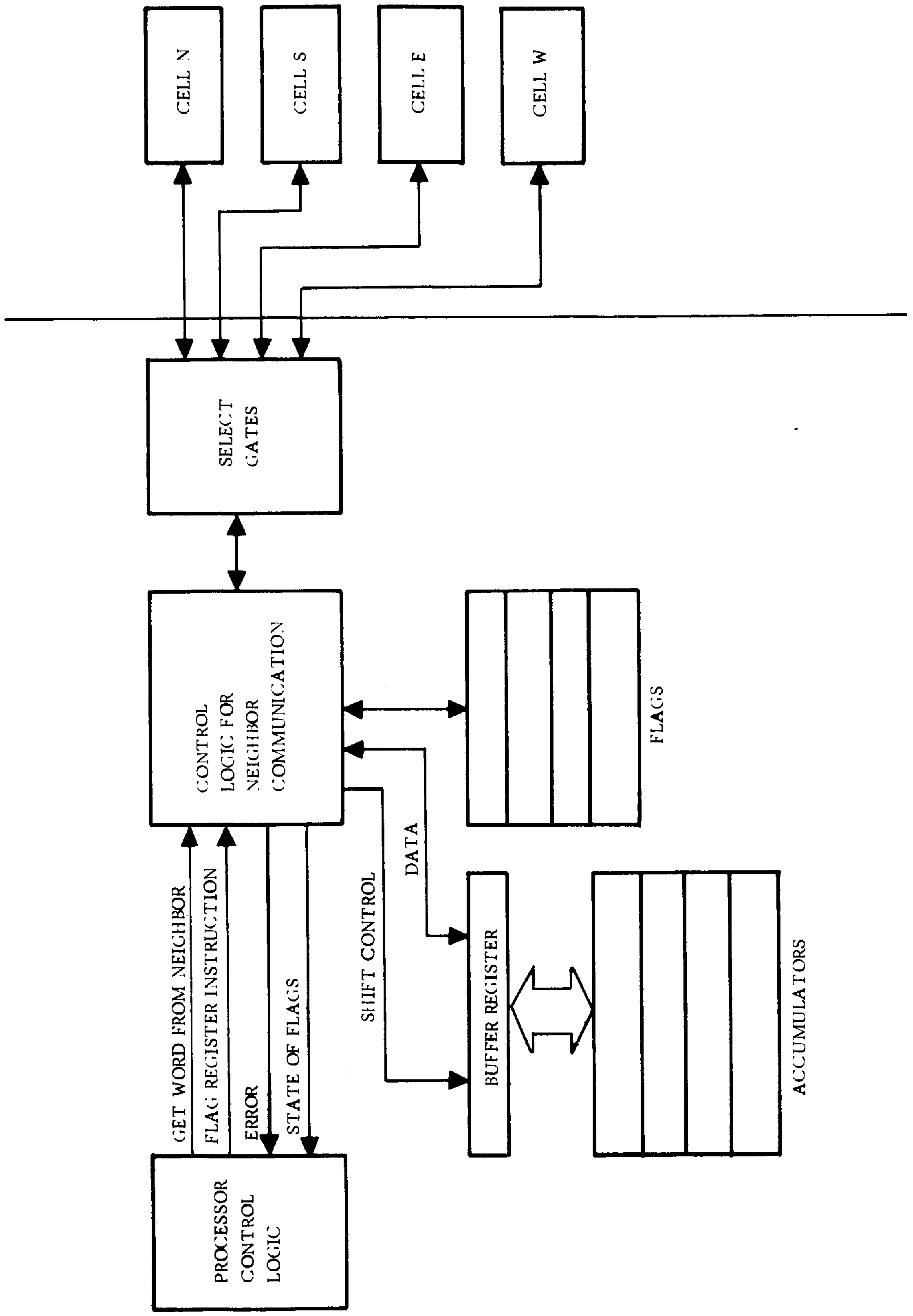


Figure 3-1. Neighbor-to-Neighbor Communication

an accumulator flagged, the accumulator contents will be transferred in parallel to the buffer register in CO. The accumulator flag is set to X. A bit-by-bit serial transfer will move the bits from CO to the buffer register in C1. The filled buffer register in C1 will be used as an operand.

Additional hardware could be added to increase speed. One buffer register is proposed, because this register must have serial shift capability. An additional buffer register would allow simultaneous processing, transmitting, and receiving. However, unless high-speed operation were required, the minimum hardware described above would be assumed.

One aspect of the control circuit should be mentioned here. A single line between any pair of cells gives the highest reliability because of the least number of connections. The only problem comes about when two cells attempt to request data from each other at the same time. One method of control is to have the cells with an even address make requests at one time, and cells with an odd address make requests during alternate times. The address assignment is done by the controller cell via the group bus. By assigning cell addresses in a checkerboard pattern, every evenly addressed cell will have four odd addressed neighbors. This problem of neighbor-to-neighbor control will be studied more in the future. The amount of clock time skew that can be compensated for will need to be determined.

3.3 SUMMARY

The proposed scheme provides one instruction to send out a word, and one instruction in the receiving cell to pick up the data and operate on it. A total of only one additional instruction compared to a sequential program is therefore required.

The high reliability is provided by having a single wire between any two cells. The flag allows a cell to test the status of a data word regarding whether it was set up and whether a request was made for the datum. Thus a malfunctioning cell that never picks up a datum will be detected by a neighbor cell who expected this cell to request the datum before some given time. The malfunctioning cell will have a low probability of causing neighboring cells programs to "hang up".

A known maximum time to transmit a word will be provided by knowing the worst case amongst the 5 cells. Since the assembler or compiler software will know the programs in all the cells, there will be no problem in timing the programs.

4. INPUT/OUTPUT

4.1 INPUT/OUTPUT OPERATION

This section of the report discusses the operation of the input/output system in the DAMP Computer. Input/output is handled by the hierarchical structure shown in Figure 4-1. This figure shows that the interface to the computer consists of serial and parallel digital lines. The conditioners C_1 through C_N each have a number of sensors connected to them. The sensors provide a variety of signals to the conditioners and the conditioners, in turn, accept these signals and provide a standard digital interface to the computer. Some devices contain their own conditioner circuitry and are connected directly into the computer; these devices will generally be connected in a full word parallel format. The bulk storage memory unit will be one of these devices: other parallel devices may include items such as buffers for video sensor data, etc..

The I/O structure described above was chosen over a completely centralized I/O structure which would absorb the conditioners into the computer and have the sensors interface directly with the computer for the reasons given below:

1. A completely centralized I/O structure is generally used to gain a more efficient hardware utilization by the consolidation of common signal conditioning functions. In this computer, reconfiguration is possible around a number of failures (down to the cell level). Since some of the I/O signals are connected directly into the cells as will be explained later, reconfiguration around a number of failures now makes a completely centralized I/O structure inefficient. This is due to the fact that all the cells would be required to have the capability for interfacing directly with any of the sensors. This approach would result in a large amount of redundant hardware and not provide an overall hardware savings.

2. The conditioner structure is easily able to adapt to a change in sensors, addition of sensors, or improvements in the sensor design. All that is necessary is to add a conditioner or replace one that is already there; whereas in the completely centralized I/O structure there is a need to redesign the cells and replace the entire computer with new chips.

3. The conditioner I/O structure also provides ease of adapting the computer system to various vehicles between missions and within missions such as a command module and a lander module of a Mars Lander Mission. These vehicles will have significantly different sensors. As a result the conditioner structure will provide the ability to use exactly the same basic computer with only the need to change the appropriate conditioners in each vehicle.

Many of the techniques that will be used in the Mars Lander Mission for handling guidance and control, status monitoring, and scientific data have been established; however, there will certainly be many new developments. As a result the sensors to be used in such a mission are presently not well defined, especially in the area of scientific experiments. This of course means that the conditioners also cannot be

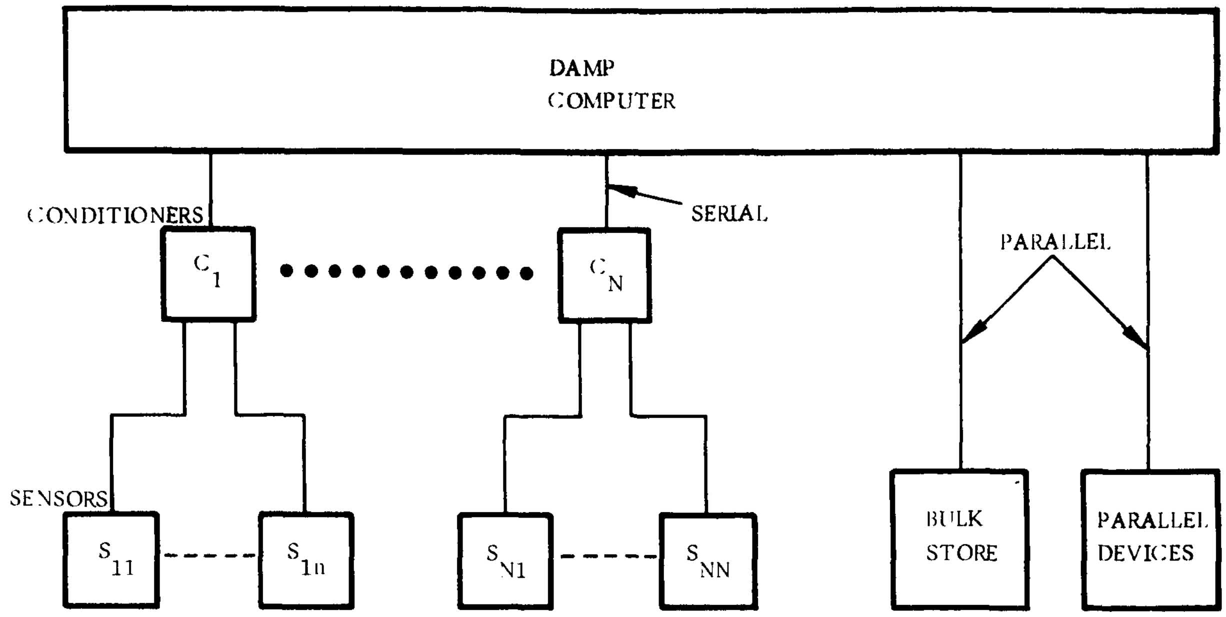


Figure 4-1. I/O Structure

well defined since their primary task is to generate control sequences, carry out analog to digital conversion, etc, for the sensors. However, certain general properties of the programs necessary to operate upon and handle the data from the sensors can be defined. These properties, typical of a wide range of spacecraft programs, will be used to obtain a first approximation to the operation of the I/O system.

The method in which the I/O ties into the block called the computer in Figure 4-1 will now be developed. A discussion of the possible methods of handling the I/O internally will be given first and then further details will be given on the selected method.

4.1.1 With the Intergroup Bus

Figure 4-2 shows the structure of the computer using an intergroup input/output scheme. The intergroup bus actually consists of two redundant half word parallel busses. Groups in the computer organization use these busses for communications amongst themselves, therefore the I/O devices attached to the busses will appear as groups functionally as far as the computer organization is concerned. The conditioners will transmit serial data to the computer; therefore, a conversion from serial to parallel is required before they get on the intergroup bus. This is accomplished in the blocks labeled I/O Cell in Figure 4-2. Two of these I/O Cells are shown in this figure, one to each of the intergroup busses, this connection provides the reconfiguration flexibility required. Conditioners handling data from critical sensors are

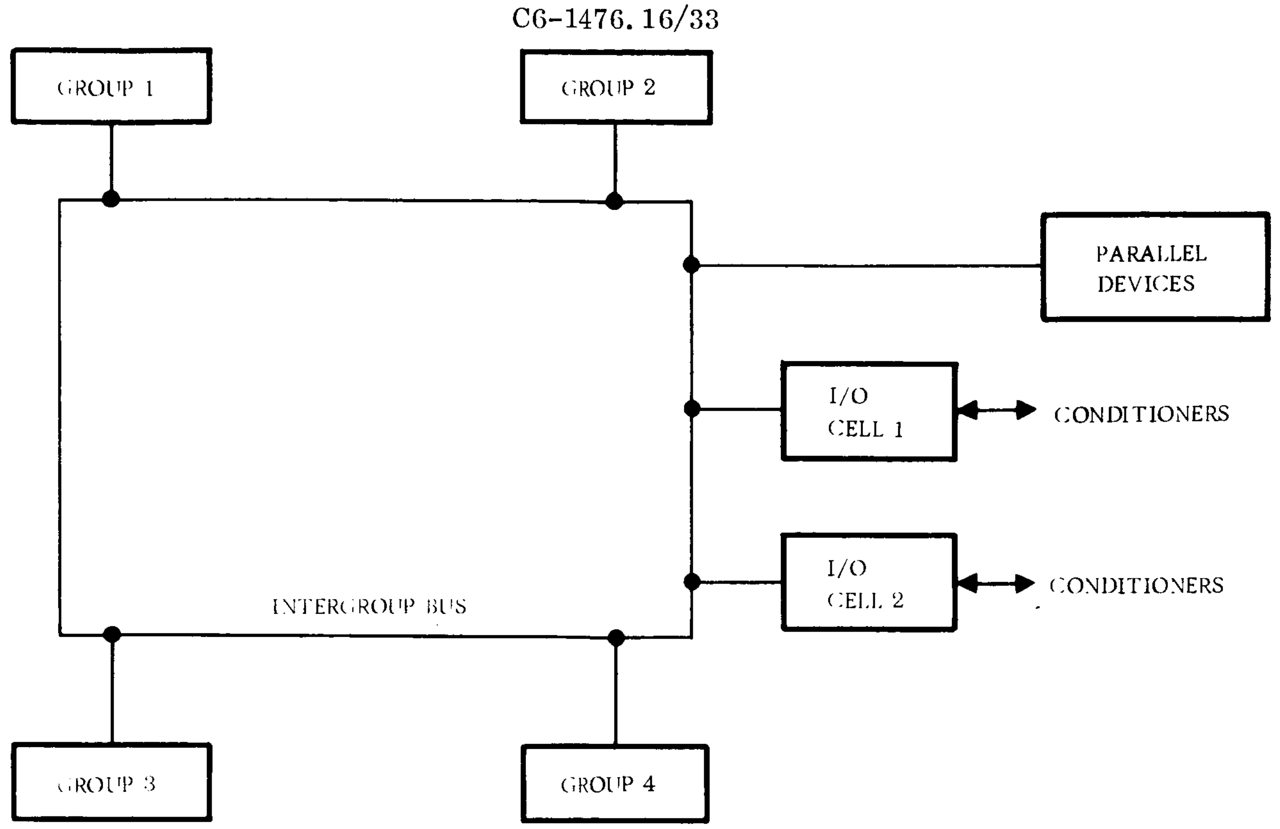


Figure 4-2. Intergroup Bus I/O Scheme

duplicated and connected redundantly into each of the I/O Cells. Therefore, if a failure occurs in the bus, I/O cell, or conditioner, the other conditioner - I/O Cell - bus connection can be used as a backup.

As noted above, sensors supplying critical data are connected to both I/O Cells; the non-critical sensors such as experiment data will be connected to only one of the I/O Cells. Two alternatives are present here: (1) to have all the non-critical sensors connected to one I/O Cell, or (2) to divide the non-critical sensors between the two I/O cells; these two schemes are shown in Figure 4-3. It should be kept in mind that Figure 4-3 shows only the sensor connected in a serial manner through conditioners to the I/O cells, there are also the parallel sensors which connect directly onto the bus. Each of the two schemes results in a different operation of the I/O system in the computer.

The first scheme where all the non-critical sensors are connected to one I/O cell results in using Bus 1 primarily and Bus 2 serves only as a backup in case of failure of Bus 1, I/O Cell 1, etc. This is due to the fact that the only I/O connections to Bus 2 are redundant connections from the critical sensors. With the second scheme both busses must be used since the non-critical sensors are divided between the two busses. The first scheme requires a higher communication rate capability on the bus when compared to the second scheme. However, note that in case of failure and reconfiguration the first scheme will be able to offer a full reconfiguration since either bus can handle the total I/O requirement. The second scheme besides reducing the communication rates required on the bus introduces a new executive method of handling the I/O. Since the I/O control and data are now divided onto two separate busses, the executive in charge of I/O has the additional task of scheduling and

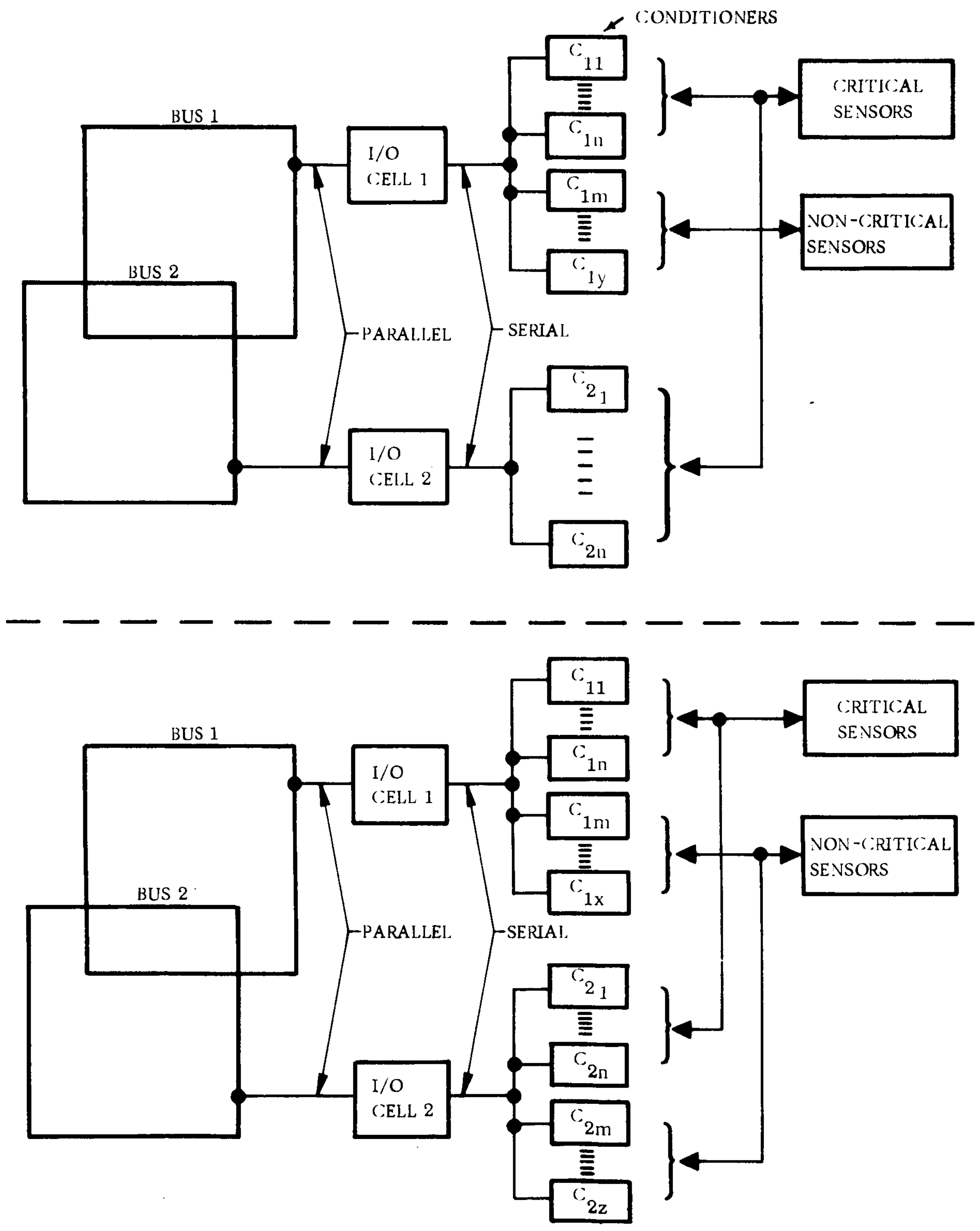


Figure 4-3. Two Methods of Intergroup Bus I/O

interleaving I/O into the groups and avoiding conflicts of simultaneous I/O on the two busses into one group. Of course the scheme with two busses also offers more flexibility in handling the I/O from an executive viewpoint since it is possible to handle periodic high priority on one bus and background type of I/O on the other bus, etc.

It should be noted here that the I/O cells are identical to the cells used in the groups in the distributed processor. The connections to the bus are now directly made through the cell and not a group switch as is the case in the groups. The serial connection to the conditioners is mechanized with the neighbor communication line. It should be noted that it is possible to use each of the four neighbor lines of a cell to sets of conditioners; this could be done to increase the communication rate capability to the conditioners (serial lines) if this became a bottleneck problem.

The cell contains processing and storage ability and can therefore function very well as an I/O processor since it has general purpose computer capability. Memory in the cell can be used to store programs for inputting/outputting data, some of these programs may be permanent and some may be loaded into the I/O cell by the executive group. The memory may also be used to store data, thereby acting as a buffer device.

4.1.2 With the Intercell Bus

In Figure 4-4 the I/O concept introduced above is extended to the intercell bus. The I/O scheme is quite similar to that given above and only the differences will be pointed out here. Only one bus is required in the group for the intercell bus, whereas two are required for the intergroup bus; therefore, only one I/O cell is shown in the figure. The sensors connected to the I/O cell may be critical sensors, non-critical sensors, or a combination of the two. Redundant inputs of critical sensors would be connected to another group, thereby providing the reconfiguration capability required if a group failed. The I/O cell is physically the same chip as the other cells in the group. It is connected on the cell bus just as the other cells are; however, its neighbor communication lines are not used with neighboring cells but for communication to the I/O conditioners. The same comments apply with regards using all four neighbor lines for I/O conditioner connections.

This approach has a communication advantage over the previous one in that the intergroup bus now is not tied up with all the I/O data since the data is now fed directly into the group where it is being used. It also does not require the overall or intergroup bus executive to be concerned with handling and scheduling the I/O; it is simply handled by the executive directly in a group. The disadvantage with the approach just presented is that the I/O cells in Figure 4-4 are somewhat specialized in comparison to the other cells in the group since the neighbor lines of the I/O cell are not connected to neighboring cells. This then makes reconfiguration difficult if the I/O Cell in the group should fail. With this approach it is also difficult to handle data which may be required by any or all the groups, e.g., data from the bulk storage device. If it is not possible to connect such devices to more than one group then a group may be required to transmit data from such devices to other groups via the inter-group bus. This obviously can place quite a burden on an intercell bus.

The logical extension of the two approaches thus far discussed would be a combination of the two: connections to the cell bus as presented above and connections to the group bus similar to that discussed previously. There are many possibilities open here. One scheme may be to connect noncritical sensors and one of the inputs from

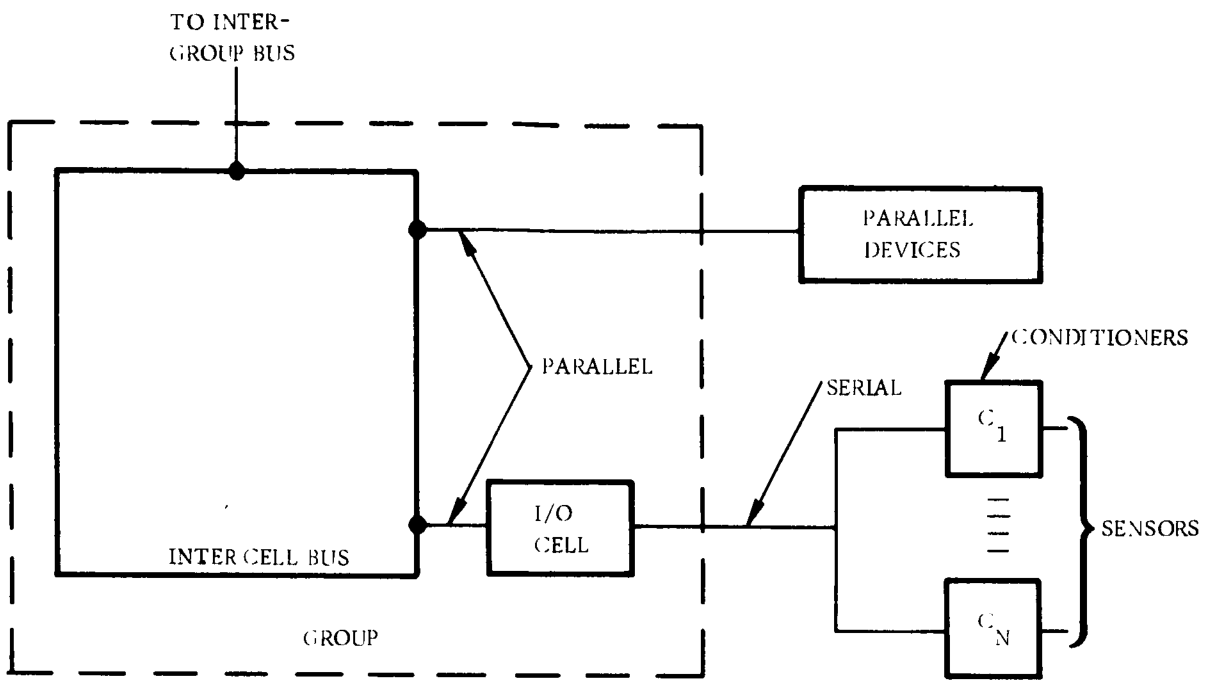


Figure 4-4. Intercell Bus I/O

critical sensors directly to groups and the other input from critical sensors and devices such as the bulk store directly to the intergroup bus. An alternative may be to connect the critical sensors directly to different groups; there are numerous possibilities here, each with certain advantages and disadvantages.

4.1.3 With the Cells Directly and the Communication Busses

Another scheme considered and the one selected is to provide connections directly to the cells in the groups for I/O and also connections to all busses in the system. This I/O approach is shown in Figure 4-5.

It should be recalled that with the previous I/O approaches of communicating to the busses; a cell had to be provided for connecting the serial conditioners into the system (I/O Cell). This cell was identical to the other cells in the system. However, it was specialized in that it was not connected in the regular pattern (array) of cells in the groups. Therefore, since the I/O cell was specialized it could not be replaced by the other cells in case of failure. The selected approach requires a separate lead to be brought out from each of the cells in the groups to an I/O connection panel; this is similar to another neighbor communication lead. Every cell therefore has the capability for handling I/O. Connections from the busses are brought out to the panel also. This scheme requires no additional or specialized I/O hardware in the system. All that is required is to provide an additional connection from each of the cells in the system (the I/O hardware can principally be thought of as connections here). Since the I/O is handled by cells directly in the group, reconfiguration around failures in terms of I/O is relatively straight forward. This is due to the fact that any of the

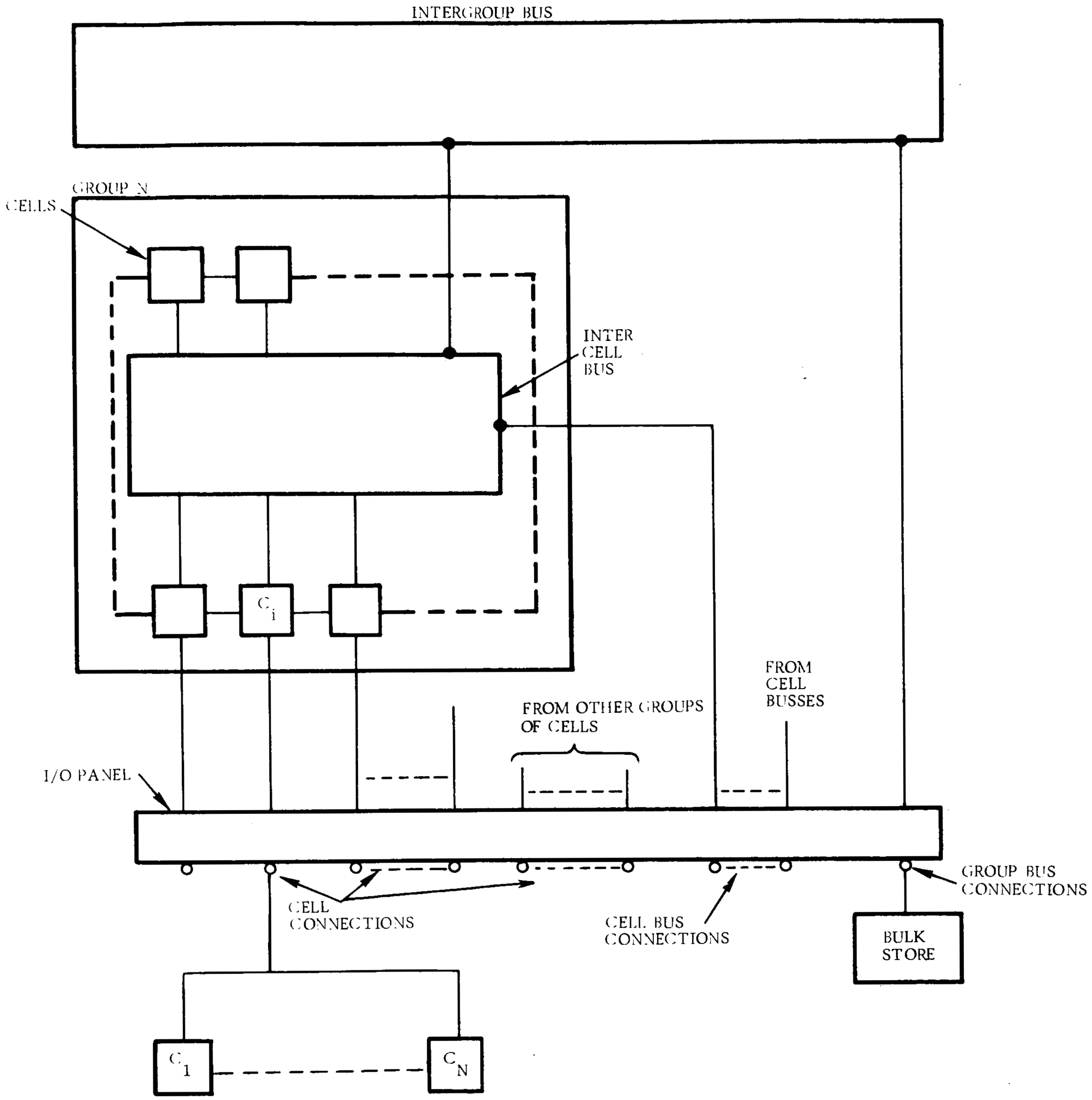


Figure 4-5. Selected I/O Approach

cells can handle I/O functions. It may be necessary to unplug - plug connectors on the I/O panel. However, a group will not necessarily then be lost due to the failure of an I/O cell.

Another possibility along these lines that was given some thought, was that of using one of the four neighbor leads for I/O as shown in Figure 4-6. In this approach connections are made to half of the neighbor lines and brought out to an I/O panel as in Figure 4-5. This eliminates the extra connection to each cell as described in Figure 4-5.

Some of the neighbor lines are now shared between neighboring cells and I/O conditioners. The problem that arises in doing this is that of avoiding conflicts of usage on the common line. If only a single line is used (bidirectional channel) for neighbor communications, then an additional connection must be added to the cells to serve as a request/acknowledge line for use of the common line between I/O and neighbors. This is required since a request/acknowledge approach must be used -- otherwise, communications over the common line would be random and meaningless. It is impossible to use only one line with a request/acknowledge approach, since the request signal from one source would naturally interfere with any established communications by the other source sharing the line. It should be noted that since it is now required to add the request line, one might just as well use this extra connection for a separate I/O line as in Figure 4-5.

If two unidirectional lines are used to mechanize the neighbor communications, the situation is somewhat different. A request/acknowledge type of approach is still required since there still would be possibilities of simultaneous usage of the communication lines (although not as probable as with one line). With a request/acknowledge approach and unidirectional lines it is possible not to require any additional lines since the scheme can be mechanized on the two unidirectional neighbor lines. This scheme would require using one of the lines for notification purposes to inhibit a request from the non using source. If two requests occurred simultaneously the proper acknowledge signal can be inhibited.

It is thus seen that if bidirectional lines are used for the neighbor communication lines, no advantage results in tying the I/O directly into the neighbor lines. However, if unidirectional lines are used, this approach could save two connections per cell.

Further explanation of the selected approach shown in Figure 4-5 will be given below. Each cell contains communication lines as shown in Figure 4-7. The I/O line shown in this figure is similar to an additional neighbor line added on to a cell. Each of the I/O lines are brought out to the I/O panel as shown in Figure 4-5. This results in an increase in the number of connections in the system, which, of course, degrades the reliability of the system. However the actual increase in terms of connections actually used is not as great as that provided (approximately 80), since only a small portion of the I/O connections to the cells will be used (approximately 10), as will be pointed out below. Therefore, this scheme actually results in only a small number of extra connections in the system.

As shown in Figure 4-5, a number of conditioners, C_1 --- C_N, are connected to a cell's (C_i) I/O line. This connection hierarchy is the same as described in the preceeding approaches. Connections are brought out from both the intercell busses

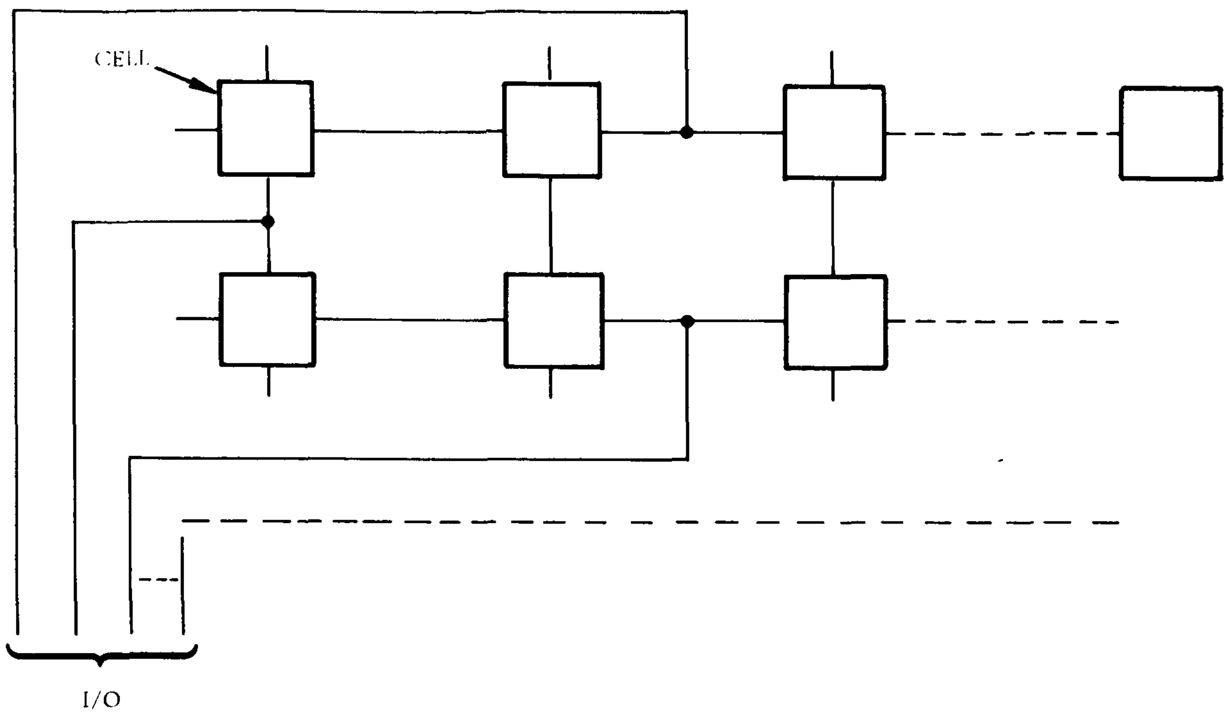


Figure 4-6. I/O Using Neighbor Lines

and the intergroup bus to the I/O panel. These are parallel connections and the type of devices that will be connected here are, e.g., the bulk storage unit, special buffers for scientific experiment data, high rate experiment sensors with large quantities of data, etc.

As mentioned previously, only a small number of the cell I/O connections will be utilized. There are a number of reasons for taking this approach. If one tended to use many of the cell I/O lines then many cells would be associated with particular conditioners or sensors. This places a severe restriction on reconfiguration. Consider reconfiguration due to phase changes for example from midcourse cruise to midcourse velocity correction; if the sensors are closely associated with particular cells one is now constrained as to where new programs may be placed in the system. It would be undesirable to have the I/O information coming into many different cells which may not even need this information and then have to be placed on the inter cell bus to cells that require it. It may be necessary to unplug-plug sensors to affect a reconfiguration in this manner.

Another disadvantage with this approach is that the reconfiguration of the system around cell failures is difficult. While the probability of a single cell failing may be quite low, there are many cells in the system (approximately 80). Therefore, by using I/O connections to many cells the probability of having a failure now associated with I/O signals is increased. If a cell fails that is being used with an I/O connection and the conditioner is connected only to this one cell, then this conditioner has to be unplugged-plugged to affect reconfiguration. The reconfiguration cannot be handled by software alone.

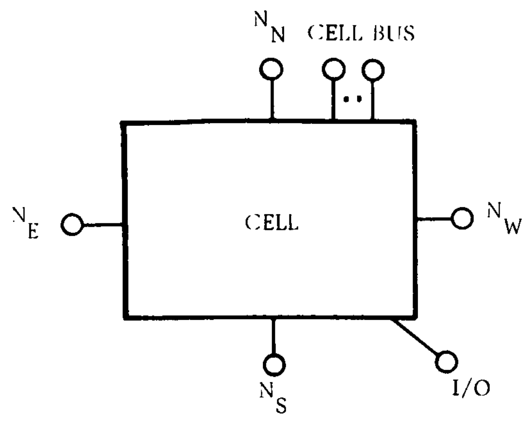


Figure 4-7. Communications Lines to a Cell

The advantage in using more connections is that more of the I/O can now be brought into cells that will use the data directly, thereby reducing the amount of I/O that has to be handled over the busses or the neighbor lines. In order to provide the reconfiguration flexibility, the approach of limiting the number of I/O connections to cells was taken.

One or more cells per group will be connected to I/O conditioners as in Figure 4-5 (typically 2 or 3 cells). The I/O control in the group may be handled in a number of ways. One approach is to have the I/O handled by both the individual cells and the cells connected to the conditioners. The individual cells will have small I/O routines for calling and accepting I/O data from the cells acting as I/O cells. The I/O cells will contain routines for servicing requests from other cells and for generating requests to other cells; they may also contain autonomous routines such as for I/O of periodic sensor data. The I/O cells will also have some memory available for buffering of I/O data.

Other possibilities include having the controller cell in the group supervise the I/O programs and/or also contain a good portion of I/O routines within itself. This may be useful for I/O data intended for more than one cell. It is also possible for the I/O cell to use its neighbor lines to pass I/O data to its four neighbors, thereby eliminating the use of the bus for certain I/O data; this could prove useful for precisely periodic sensor data. The intent here is to present the general concept of the I/O system; further details as to the operation of the I/O cells will be gone into in later phases of the study when the executive design is attempted.

As mentioned previously, connections are provided to the busses. Devices connected to the intergroup bus such as the bulk storage unit will effectively function as groups. The overall executive will control the communication (I/O) between such devices and the groups in the system. Devices connected to the intercell bus will effectively function as cells; the controller cell in the group may control I/O to such devices and/or the individual cells may also provide requests for I/O action to such devices.

One other point should be mentioned here, critical sensors will be connected to I/O cells in two different groups. This will provide for the required reconfiguration capability if the intercell bus or I/O cell that a critical conditioner is connected to should fail.

4.2 I/O MECHANIZATION

A preliminary design of the I/O system was completed to get an estimate of the I/O mechanization requirements. This section presents a summary of this investigation.

The I/O cells are to carry out their communications (both in and out) with the conditioners over a single line connected between the I/O cell and all conditioners that are to communicate with the cell. The I/O cell has control over this line and operates under control of an internally stored program.

The operation of the I/O system will be explained below. Assume that the I/O cell desires to output a set of words to a conditioner. The following instruction is executed:

I/O	Address

The address specified in the I/O instruction is used to access the I/O control word:

bits:	1	7	8
	Rd/Write	Conditioner/Device	Word Count

This control word tells whether this is to be an input or an output operation (Rd/Write), which conditioner and device the operation is to be carried out with, and the number of words to be communicated. Provision is made to transmit up to 256 words. If this is more than needed and 128 proves adequate, one can substitute an indirect bit in place of one of the word count bits; this indirect bit could be used to aid in locating the address of the first word to be communicated as will be explained below.

Seven bits have been allotted for identification of the conditioner/devices. This provides the capability of handling up to 128 devices per I/O cell. Again, if this proves too few devices, one may possibly use one of the word count bits for conditioner/device identification. A preliminary feeling for the breakdown is to provide for 8 conditioners and 16 devices per I/O cell.

The sequence of events occurring in executing the I/O operation will now be explained. A combination of hardware and software will be used and the relative usage of each to accomplish the instruction may be varied; the description given is for a preliminary description only.

Two possibilities are available for locating the first word in the set of words to be transmitted. The word may be located in the location immediately following the control word described above, or it may be located indirectly by using the address contained in this location to specify the location of the first word. Another possibility is to use an indirect bit in the control word, thereby offering the potential of providing both possibilities described above.

The following sequence of operations are carried out to execute the I/O operation:

- Fetch control word
- Place proper bits in buffer register
- Shift out buffer register over I/O line
- Place word count in a certain location in memory
- Fetch address in location following control word
- Place address in a certain index/bank register
- Transfer to the input or output (depending on rd/write bit) software routine

The address of the input/output routine could be hardwired or previously setup in a certain index/bank register. As noted above hardware and/or software may be used to execute the I/O operation described above. A preliminary feeling is that the above may be carried out most efficiently by hardware.

The buffer register now contains the following information in it:

bits:	1	1	7	8	1
	Sync	Cont/data	Cond/Device	Spare	Rd/Write

The buffer register is seen to be 18 bits in length. This is due to the fact that two control bits must be added to utilize only one line in the I/O mechanization; these are the sync and control/data bits. The sync bit is always a one and is used to synchronize the conditioners. The control/data bit identifies the following 16 bits as either a control word or a data word. The control word shown for the conditioners uses 7 bits for the conditioner/device identification--8 bits are not used (actually the word count may be placed in here if this is deemed useful to the conditioners) and one bit for identification of the operation as input or output.

The conditioners are clocked with the I/O cell and utilize the sync bit to set a counter. Upon counting to 18 the conditioners reset and are ready for the next transmitted word. Upon detection of a word identified as a control word, each conditioner will examine the conditioner/device bits to determine if the control word is for this particular conditioner. If it is then this conditioner will lock on to use the I/O line for receiving or transmitting the data that follows.

A preliminary description of the output routine will be given below. This routine will be described as a software approach. It should be kept in mind that hardware may be substituted to achieve a faster execution if this is found to be necessary.

Output routine:

- Load Accum. with first word (address specified by previously setup index/bank reg.)
- Transfer Accum. to Buffer Register (with this the buffer is automatically started shifting out)
- Load Accum. with word count
- Subtract 1
- Test and transfer on 0
- Modify index/bank reg. for 1st instruction
- = 0 Jump to some location back in program (location set before I/O instruction executed)
- Jump to master I/O routine

To get back to the output routine (when the buffer register is shifted out), an interrupt is sent from a counter associated with the buffer register; this transfers the program to the output routine (location of this routine could be a hardwired location or previously loaded in an index/bank register).

Input Routine:

- Transfer Buffer to Accumulator
- Store Accum. in first word (location setup by index/bank reg)
- Load Accum. with word count
- Subtract 1
- Test and transfer on 0
- Modify index/bank register used in 2nd instr.
- = 0 Jump to some location back in program
- Jump to master I/O routine

To get back to the input routine (when the buffer register is full again), an interrupt is sent from the same counter associated with the buffer register (the I/O instruction sets a mode flip flop to determine if the input or output routine is to be entered).

As noted previously this description is preliminary, further details will be examined in later phases of the study.

The above description results in an I/O system completely under control of the I/O cell. The conditioners cannot interrupt or request I/O operation independently from the I/O cell. To facilitate requests from devices for I/O operations (for example a request from the astronauts panel or a buffer holding experiment data), it is possible to insert in each (or possibly only several) conditioners a request register that may be sampled periodically by an input operation by the I/O cell. Then the I/O cell may decide how to handle the requests, if any.

Another possibility is to add a separate request line as shown in Figure 4-8. This line passes serially through the conditioners. Periodically, the I/O cell sends out a request pulse. If no requests are present in the conditioner, the connection will be successively completed to the last conditioner where the request line will be grounded. This, then, signifies no request is present. The first conditioner with a request that receives the pulse will not complete the circuit. The I/O cell recognizing this knows a request is present but not from which conditioner. The conditioner with the request that received the pulse will now send a control word to the I/O cell indicating what I/O action is desired with the proper identification.

The advantage with this scheme is that the conditioners may be sampled very quickly and easily (minimum software in the I/O cell) by the I/O cell to handle requests by the devices for I/O operation. Its disadvantage is that it costs an extra connection. Note however that if a failure occurred in this connection one could always revert back to the original approach without this scheme.

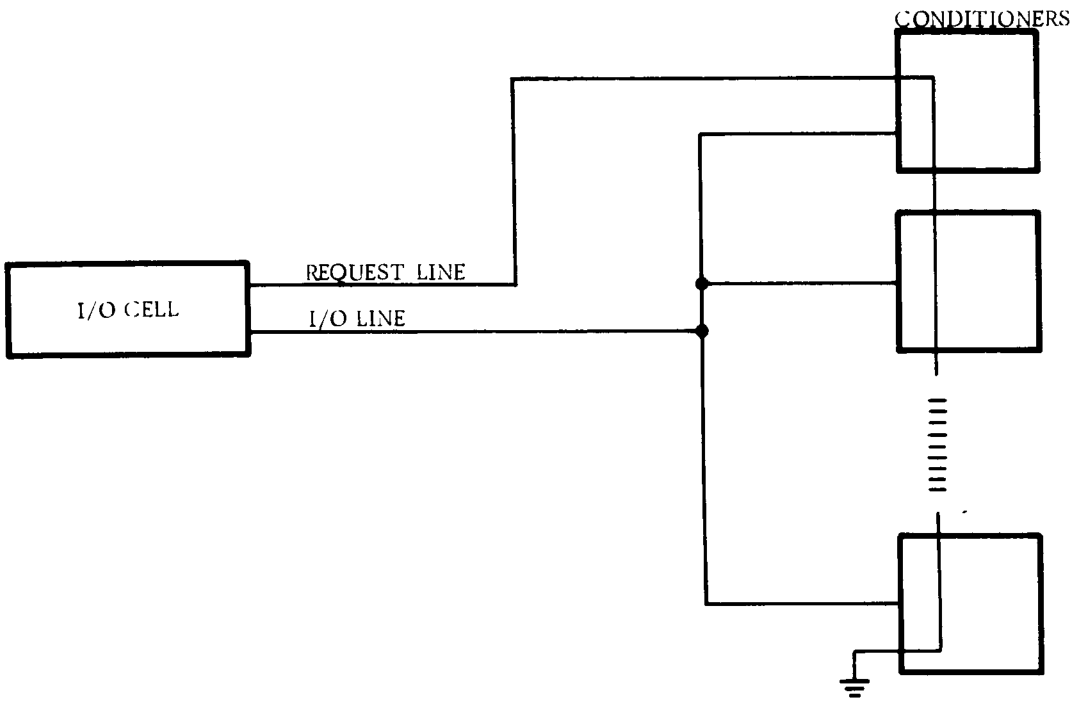


Figure 4-8. Request I/O System .

5. GROUP ARCHITECTURE

5.1 INTRODUCTION

Many different computer systems have been studied during the course of this study. A description of these systems appeared in Section 5 of the previous quarterly report (ref. 1). The distributed array memory and processor system was found to be the most useful for the general computations needed on future spaceflights. This distributed system, shown in Figure 1-1, requires a unique architecture to make a capable and reliable system.

Architecture means the combining of software and hardware features to make a balanced useful system that will meet the requirements set upon the computing system. Some of the considerations, such as memory size and approximate processor capability, are based upon the ground rule to build a cell upon a single wafer. This section describing the group architecture will describe the features desirable to unify the cells into a working group.

The distributed processor system consists of groups, which are made up of cells. Because the cells in a group are connected by neighbor communication lines, and the controller cell can send global instructions to cells, the group is the fundamental unit of the computing system. The software studies to date indicate the compiler must be aware of the cell memory contents, the cell bus loading, and the controller cell capabilities when compiling programs. For these reasons the architectural studies were applied to the group.

The features and characteristics of a group are described here. All these features may not be needed; future studies will determine the useful features to be retained, and the features of little value to be discarded.

5.2 CELL STATES

A fundamental ground rule in this study has been to make all cells of identical hardware. When the cells are operating, the cells function in one of seven states, shown in Table 5-1. Although all the cells are identical in hardware, a cell always exists functionally in one of seven different and mutually exclusive states.

A permanently failed cell is placed in state 1 by a combination of software and hardware controls. These cells will not be used again. The reconfiguration studies will determine the software and hardware required to diagnose and shut down a malfunctioning cell.

State 2 is the power saving state for cells that are not needed presently. If standby power is applied to the level register and the cell bus gates, the main power to this cell may be turned on by the controller cell and switched to another state. The controller cell then may reload the cell's memory. If all the power has been switched off, a special restart procedure, using the neighbor communication lines, must be used. This problem will be studied as part of the reconfiguration studies.

Table 5-1. Cell States

1.	Permanently failed - power off
2.	Shut down - power saving state
3.	Independent
4.	Dependent under global control (Global State)
5.	Dependent under local control
6.	Dependent in wait state
7.	Controller cell

Independent cells are functionally similar to a conventional computer. These cells fetch all instructions and operands from their memories. The cell that is in the independent state stays in this state until the controller cell sends a command on the intercell bus with a cell address equal to the contents of the cell's identification (ID) register. This command can cause this cell to change states. Each independent cell must be addressed individually. The independent cells can process problems that are not amenable to global processing.

Dependent cells respond to global instructions and global level commands sent out from the controller cell. A dependent cell exists in one of the states 4, 5 or 6. Which of the three depends upon the level of instructions being sent from the controller cell and the cell's level register contents. The concept of levels is described later under cell identification.

A dependent cell in the global state (also called the active state) is receiving instructions from the controller cell via the cell bus.

A dependent cell that is not at the proper level to receive global instructions can idle and not execute instructions. This is the wait state. If the controller cell is servicing certain dependent cells, other dependent cells may wait their turn for service.

A dependent cell, instead of waiting for the controller cell to send the instructions for its level, may fetch and execute instructions from its own memory. This is the local control state.

The concept of having both independent cells and dependent cells in a computer system is an important concept developed in this study. Other studies of similar computer systems require all cells to be independent or all dependent. With this improved system, the system's problems may be solved most efficiently by using both independent and dependent cells.

The use of local control by dependent cells means that the cell bus is not wasted sending instructions when the instructions could be better stored in the cell's memory. With this feature, the cells can efficiently use local programs to correct for bad data and handle exceptional conditions. The cell can enter the local control state, do some processing, and later inform the controller cell of the situation.

The seventh state of a cell is the controller state. In this state, a cell may issue global instructions and control the cell bus. The fundamental ground rule of making all the cells of the same hardware allows any cell to become a controller cell. This gives the advantage that the controller cell functions may be switched among several cells. Thus there is no requirement that all the executive and controller programs fit in one cell.

There is only one controller cell in a group. The reason is the controller cell controls the cell bus, and two cells cannot be allowed to issue conflicting commands. Software and hardware interlocks will be used to insure only one controller cell is in a group.

The group switch, shown in Figure 1-1, is part of the group although it is not a cell. The group switch, like a cell, has an ID register. The group switch responds to control words containing the proper ID bits. The group switch will perform the operation given in the control word (CW). Thus the controller cell will operate the group switch.

5.3 CELL IDENTIFICATION

The distributed computer system has a central source of instructions which are sent to many cells. Instructions may be fetched from a cell's memory. The DAMP system divides the cells into eight groups, or levels. Each cell in a group has the same level number. In addition, each cell is given an identifier, also known as the cell address. Thus a cell has two "names", a common first name (level) and a unique last name (identifier). This concept of having two names is important when discussing the dependent and independent cells.

Independent cells use only one name, their identifier or cell address. The level (or first name) is not used, and, although present in a level register, has no meaning.

Dependent cells use two names. The controller cell may send out a first name (level number) to all the dependent cells. All the dependent cells at this level will respond. If a last name (cell address) is sent, only the cell with this name will respond.

The instructions sent by the controller cell follow the name. The cells that responded to the name will receive the instructions that follow the name. For example, assume a system with 7 cells as follows:

First Name	Last Name	Dependent	Independent
JOE	SCOTT	x	
BOB	ROSE	x	
HELEN	TRUMP		x
BOB	MILLER	x	
BOB	JOHNSON	x	
JOE	SMITH	x	
HELEN	DAVIS	x	

The controller cell sends the following instruction groups. The results are explained below.

JOE: Load X, Store Y, BOB: Load A, Add X, Subtract B, Store Y, HELEN: Load A, Store Y, ROSE: Add M, Add N, ...

Two cells (JOE) will Load X, Store Y. Three cells (BOB) will execute the next four instructions. One cell will execute the next two instructions. (The cell TRUMP is an independent cell and does not respond to first names.) The name ROSE is a last name, thus only one cell will execute these instructions.

5.4 SOURCE OF INSTRUCTIONS

The traditional computer has instructions stored in a memory which is always available to the processor. The processor controls the instruction fetch sequence by using the program counter. In most modern machines, the instructions are located in a random access core memory, and the program counter is incremented to fetch sequential instructions. A jump is performed by loading the program counter with the address of the next desired instruction.

The independent cell receives all of its instructions from the cell's memory, like the traditional computer. The program counter is used to control the fetch of instructions.

The dependent global cell gets its instructions from the controller cell. The cells receive the instructions from the cell bus and then execute them. The controller cell precedes the instructions with a name. The level number (name) is contained in a control word sent on the intercell bus. This control word is a prefix to a group of instructions (including their modifiers). This prefix is the level of all instructions until a new level prefix is sent or other control instruction is sent.

Every dependent cell compares the level prefix sent by the controller cell to the level register contents contained in the cell. If the prefix and the level register contents are different, the cell ignores all the instructions, data, etc. sent by the controller cell, until a new level prefix (or other control word) is put on the bus.

An example is given in Figure 5-1. Remember that every cell is required to examine every control word, but will not perform the control word operation if the cell is at a different level, or has the wrong ID (cell address).

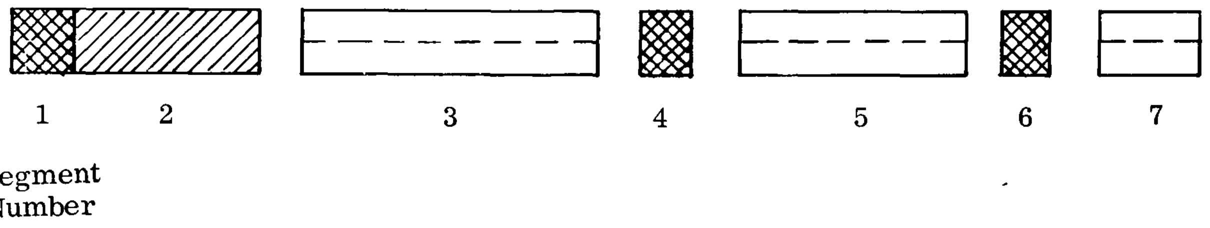


Figure 5-1. Bus Operation Example

▩ : Control Byte (CB) All cells will examine this byte. If the cell matches this CB, the cell will receive the control word.

▨ : Control Word (CW) This word includes the CB, and defines an operation to be performed by the cell. Often the CW consists of only a CB.

⬚ : Data. In this example, we shall assume that the Control Word specified that instructions are contained here.

When segment 1 in the example occurs, all the cells in the system will examine the CB. We will assume the CB is a type that specifies a level. Thus all the dependent cells <u>at this level</u> will be ready to receive the CW (segment 2) and are automatically placed in the dependent active (global) state. These cells in the global state will receive the CW (segment 2) and will receive the instructions and data following (segment 3). No other cells will receive any instructions or data (segment 3) from the bus until the next CB occurs (segment 4 in the example).

When segment 4 comes on the bus, all the cells will again examine the control byte. In the example, it shall be assumed the CB specifies a different level. The following actions will occur:

In cells that were active, the CB at a new level will set these cells to wait state.

In cells that were not active, and are at the new level indicated in the new CB (segment 4), these cells will become active and will receive the data (instructions) following (segment 5). All other cells are left unchanged.

Thus it can be seen that many sequences of global instructions may be sent to many sets of cells at a very low overhead cost to switch between sets. The low overhead is advantageous when many cells are at each level and short sequences of instructions are to be transmitted to each. Also the bus is used very efficiently.

To summarize, a dependent active (or global) cell is a cell that is receiving global instructions and data. By definition, a global cell is at the same level as the global instructions. Actually, the level is in the prefix CB, there is no level transmitted with each instruction. The term "global instructions level" will refer to this prefix, although the term is not exactly correct.

The dependent cell not receiving instructions from the inter-cell bus may fetch instructions from its own memory. This cell is in the dependent local control state.

The controller cell always fetches instructions from its own memory. The instructions destined to be executed by the dependent global cells are not executed by the controller cell. All other instructions are executed by the controller cell and are not sent to the global cells. This is explained further in the section describing the controller cell instruction execution.

5.5 SOURCES OF ADDRESSES

The computer technology has developed over the years many ways of specifying a memory address. The early machines had the operand address given in the instruction. Later, index registers were used to modify the instruction address. Memory banks were used to save instruction bits. The traditional computer had several ways of determining the final (or effective) address that was used to address memory.

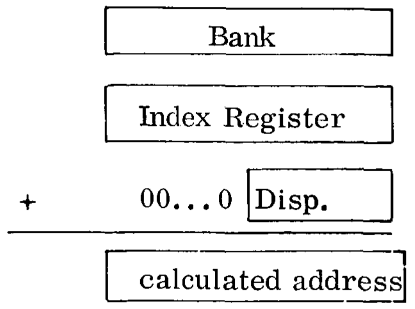
The cells in the distributed processor computer also have several ways to specify an address. All the ways will be described here, although some are not used by cells in certain states.

The address may be specified by adding the instruction displacement and the bank register (also known as a base register). The bank register is 16 bits long.

```
           | Bank              |

   +         00...0 | Disp.    |
  ------------------------------
           | calculated address|
```

This sum is called here the calculated address. If an index register is specified, it is also added.



These two calculated addresses use the registers located in the cell.

Independent cells will obtain all the parameters that make up the address from the cell itself; dependent global cells will obtain the displacement from the instruction that was sent on the cell bus. The bank and index register are always from the cell.

In addition to the calculated address, a new concept of a given address is used. A given address is an address that is used instead of the calculated address.

A dependent global cell recognizes a given address by a special control instruction received on the cell bus. This special instruction is called a GC format instruction. The format instruction is really an 8-bit byte sent from the controller cell to signal the global cells that a given address, in addition to the instruction, is to be sent on the cell bus. The sequence is as follows.

Time	Contents of Cell Bus	Length
↓	GC Format-address is given	(8 bits)
	Instruction	(16 bits)
	Given address	(16 bits)
	.	
	.	
	.	
	subsequent instructions	
	.	
	.	
	.	

The global cell normally expects to receive 16-bit instructions. However, this normal sequence is altered by a format instruction. This instruction is a control byte and tells the dependent global cells that something new has been added. In this case, that a 16-bit address follows the next 16-bit instruction. The global cell will execute the instruction using the given address instead of the calculated address. Thus the controller cell may send an address to all the global cells instead of having the cells calculate the address.

The independent cells (and the dependent cells under local control) may use the format instruction. In this case, the format instruction is located in the cell's memory and is fetched as any other instruction. After the format instruction is executed, the processor knows the type of data contained in the following memory locations. An example is given here.

Location	Contents	Length
START	Format GC Instruction-address is given	16 bits
+1	Instruction	16 bits
+2	Given Address	16 bits
	. . .	
	subsequent instructions	
	. . .	

Here, the instruction at START +1 is executed using the given address instead of the calculated address. The use of the GC format instruction is called instruction modification. The modification is usually not a change in the operation of the instruction, but rather a respecification of the address.

5.6 SOURCES OF DATA

The cell in the distributed processor computer system can obtain data from many sources. Some sources are available to all cells irregardless of their state, others are available only to cells in a particular state.

All cells have access to data stored in their memory. The present cell concepts has no division of memory into data areas, read-only areas, etc. Thus any location in a cell is available to the processor.

All cells may obtain data from their neighbors. Because the neighbor to neighbor data transfer is independent of the cell state, the neighbor communication system is described separately in Section 3.

Cells may receive data from outside the group via the cells I/O line. This system is described in the section on Input/Output Operation (Section 4.1).

Cells may receive data from outside the group or from other cells via the inter cell bus. This system of a cell communicating directly with another cell via the inter-cell bus is described in the section on the communication bus operation (Section 6.2). By the same system, when one "cell" is the group switch, a cell can receive data from the outside world such as other groups and the bulk storage unit.

Dependent global cells may receive data from the controller cell. A format instruction is used to indicate to the receiving cells that data is being transmitted in addition to instructions. The format instruction is used like the GC format instruction described in the preceding section, Sources of Addresses.

A 16-bit data word may be sent to global cells by sending the following sequence on the inter-cell bus. The GC format instruction is called a D16 format.

Time	Contents of Cell Bus	Length
↓	Format - 16 bit data follows	8 bits
	Instruction	16 bits
	Data	16 bits
	.	
	.	
	.	
	subsequent instructions	
	.	
	.	
	.	

This format instruction indicates an instruction is followed by a data word of 16 bits. The instruction is executed by the cell. The operand used, however, will be the data word received from the inter-cell bus and not the data word usually fetched from memory. More details concerning the operands and data are given in the section on instruction execution (Section 5.7).

Having data sent by the controller cell means that the individual cells do not each have to store constants. To have 20 cells all store pi, e, and other constants is an inefficient use of cell memory, whereas the controller cell has to store the constant but once and send it out when it is needed. The constants are sent at the time they are used; thus they need not be saved in the cells memory.

A 32-bit data word may be sent to dependent global cells. The GC format instruction is now called a D32 format.

Time	Contents of Cell Bus	Length
↓	Format - 32 bit data follows	8 bits
	Instruction	16 bits
	Data	32 bits
	.	
	.	
	subsequent instructions	
	.	
	.	

This is similar to the D16 format. The instruction will use the 32-bit data word in performing its operation.

Another format instruction is called the I (for Immediate) format Here the data is the displacement field in the instruction Naturally, the magnitude that may be sent depends upon the length of the displacement field in the instruction. The I format is especially useful when loading registers with small values. The I format is received by a dependent global cell as shown on the next page.

Time	Contents of Cell Bus	Length
↓	Format-Immediate	8 bits
	Instruction	16 bits
	.	
	.	
	subsequent instructions	
	.	
	.	

For example, if the instruction is a Load Index Register 3, the displacement field of the instruction, preceded by zeros, will be loaded into index register 3.

The last format instruction that concerns data is the DS format. This is a very special format whose usefulness is yet to be determined. It was designed to rapidly move data from the controller cell to a group of cells or a cell. The sequence received by a dependent cell is as follows:

The last format instruction that concerns data is the DS format. This is a very special format whose usefulness is yet to be determined. It was designed to move rapidly data from the controller cell to a group of cells or a cell. The sequence received by a dependent cell is as follows:

Time	Instruction and Data	Length
↓	GC format - DS	8 bits
	Instruction	16 bits
	address	16 bits
	data word 1	16 bits
	data word 2	16 bits
	data word 3	16 bits
	⋮	
	data word N	16 bits
	GC format-End of DS	8 bits

The receiving cell will receive the DS format instruction. The instruction following will be executed using the given address and the first data word. The given address will be incremented by one, and the instruction will be repeated using the second data word. The operation will continue until the GC format byte. End of DS, is received instead of a data word at N+1. The DS is seen to be useful if the instruction is a store or compare to memory type of instruction.

The above description of instruction modifiers to allow the controller cell to send data to the dependent cells applies to dependent global cells. The instruction modifiers may also be used by cells in other states. Of course, the format instructions must be stored in the cell's memory, and are not sent over the intercell bus. Section 5. 7 describing the instruction execution should be consulted for more details.

An example of how GC format instructions can be stored in a cell's memory and used to modify instructions is given below.

Location	Contents	Length
START	GC Format-D16 data follows	16 bits
+1	Instruction - Load Acc 1	16 bits
+2	data word 1	16 bits
+3	Instruction - Load Acc 2	16 bits
+4	.	
	.	
	.	
	subsequent instructions	
	.	
	.	
	.	

The GC instruction at START indicates to the processor that the next instruction is followed by a word of data. The load accumulator 1 instruction will load accumulator 1 not with the contents of the memory location specified by the calculated address but with data word 1 located at START +2. It is seen the GC is used here to respecify the location of the data to be loaded into the accumulator.

The instruction at START +3, because it is unmodified, is executed in a normal manner.

MOST instructions may be modified by a GC format instruction. Table 5-3 gives a list of all the instruction types and how they are affected by modification.

5.7 EXECUTION OF INSTRUCTIONS

The instruction execution in the distributed processor system is a complex subject. The execution depends upon the state of the cells and upon where the addresses, instructions and data are located. To simplify the explanation and to delete many small details, a general computer organization has been assumed. The general principals discussed here will be the same no matter how the final hardware design changes from the present concepts.

The processor section of the computer is assumed to contain the program counter, instruction decoding logic, adders and several registers. The registers are accumulators, index registers, and base registers. The registers may be located in an addressable section of the cells memory, or they may not be addressable by the programmer. The cell also contains an identification register and a level register.

The instructions have been divided into several general categories (Table 5-2). All the instructions in a category are executed in a similar manner. There will be a description for each instruction category for the different cell states. Table 5-3 summarizes the instruction execution.

Table 5-2. Instruction Categories

1.	LR	Load Register from a memory location
2.	STR	Store Register into a memory location
3.	OPR	An operation is performed between a register and a memory location contents, the results are in a register.
4.	RR	An operation is performed between one register and another register.
5.	R	Single register operation, such as shift.
6.	EXEC	Execute an instruction in a memory location.
7.	COMP	Compare the contents of a memory location (or register) with a register. The results of the comparison are saved in the COMPARISON flip-flops.
8.	SKIP	Test the contents of a memory location (or register) with a register or implied value. The result is true or false.
9.	JUMP	A new sequence of instruction is begun. The jump may be combined with a test to make a conditional jump.
10.	CC	Controller Cell instruction. The instructions and commands used by the controller cell, excluding Global Control instructions.
11.	GC	Global Control instructions. These instructions control the levels and dependent cell execution of global instructions.
12.	IO	Input-Output instructions. These instructions initiate and control I/O operations.

5.7.1 Dependent Global Cell

A dependent cell may receive instructions, data, and commands from the cell bus. The global cell, or active cell, is receiving instructions and executing them as they are received; the level prefix placed before the instructions by the controller cell is the same as the contents of the level register in the global cell.

Although the global cell receives instructions from the intercell bus, the registers, addresses, and data are usually from the cell's memory. Thus several global cells will receive the same instruction, but all may use different addresses and process different data. The exceptions are indicated by the use of a GC Format modifier byte preceding the instruction. This concept was explained in the previous sections.

1. LR instructions are all instructions that fetch an operand from a memory location and load the contents into a register. The address of the memory location is calculated by adding the base register, index register (if one is specified) and the displacement from the instruction. Only the displacement is received on the intercell bus. The low-order nine bits are used to

Table 5-3. Summary of Instruction Execution

Instruction Category	Cell State	Source of Instruction	Source of Address	Operand 1	Operand 2	Notes
LR Load Register	DG	Cell Bus	Calc (G-Cell Bus)	M (D16, D32, I - Cell Bus)	R2	Operand 1 is loaded into the specified register
	DL	M	Calc (G-M)	M (D16, D32, I - M)	R_2	Operand 1 is loaded into the specified register
	Ind	M	Calc (G-M)	M (D16, D32, I - M)	R_2	Operand 1 is loaded into the specified register
	CC	M	Calc (G-M)	M (D16, D32, I - M)	R_2	Operand 1 is loaded into the specified register; this instruction is normally transmitted and not executed by the CC.
STR Store Register	DG	Cell Bus	Calc (G-Cell Bus)	M	R_2 (D16, D32, DS-Cell Bus)	Operand 2 is stored in memory location M
	DL	M	Calc (G-M)	M	R_2 (D16, D32, DS-M)	Operand 2 is stored in memory location M
	Ind	M	Calc (G-M)	M	R_2 (D16, D32, DS-M)	Operand 2 is stored in memory location M
	CC	M	Calc (G-M)	M	R_2 (D16, D32, DS-M)	Operand 2 is stored in memory location M. This instruction is normally transmitted and not executed by the CC.
OPR Operation Performed Between Memory and Register.	DG	Cell Bus	Calc (G-Cell Bus)	M (D16, D32, I, DS-Cell Bus)	R_2	The operation is performed between operand 1 and operand 2
	DL	M	Calc (G-M)	M (D16, D32, I, DS-M)	R_2	The operation is performed between operand 1 and operand 2
	Ind	M	Calc (G-M)	M (D16, D32, I, DS-M)	R_2	The operation is performed between operand 1 and operand 2
	CC	M	Calc (G-M)	M (D16, D32, I, DS-M)	R_2	The operation is performed between operand 1 and operand 2, this instruction is normally transmitted and not executed by the CC.
RR Register-to-Register	DG	Cell Bus	-	R_1	R2	Register to Register instructions.
	DL&Ind	M	-	R_1	R2	Register to Register instructions
	CC	M	-	R_1	R2	Register to Register instructions are normally transmitted and not executed by the CC.
R Register	DG	Cell Bus	-	-	R_2	Single register instructions
	DL&Ind	M	-	-	R_2	Single register instructions
	CC	M	-	-	R_2	Single register instructions are normally transmitted and not executed by the CC.
EXEC Execute	DG	Cell Bus	Calc (G-Cell Bus)	M	-	The instruction fetched from M is executed
	DL&Ind	M	Calc (G-M)	M	-	The instruction fetched from M is executed
	CC	M	Calc (G-M)	M	-	An execute instruction is normally transmitted and not executed by the CC.
COMP Compare	DG	Cell Bus	Calc (G-Cell Bus)	M or R_1	R_2 (D16, D32, I, DS-Cell Bus)	The level register may be changed by a compare, the compare flip-flops are always set.
	DL	M	Calc (G-M)	M or R_1	R_2 (D16, D32, I, DS-M)	The level register may be changed by a compare, the compare flip-flops are always set.

Table 5-3. (Cont)

Instruction Category	Cell State	Source of Instruction	Source of Address	Operand 1	Operand 2	Notes
COMP Compare (Cont)	Ind	M	Calc (G-M)	M or R_1	R_2 (D16, D32, I, DS-M)	A compare may cause a jump, the compare flip-flops are always set.
	CC	M	Calc (G-M)	M or R_1	R_2 (D16, D32, I, DS-M)	A jump may occur if this instruction is executed. Compare instructions are normally transmitted and not executed by the CC.
SKIP Test and Skip	DG	Cell Bus	Calc (G-Cell Bus)	M or R_1	R_2 (D16, D32, I, DS-Cell Bus)	The level register is incremented if the test is true.
	DL	M	Calc (G-M)	M or R_1	R_2 (D16, D32, I, DS-M)	The level register is incremented if the test is true
	Ind	M	Calc (G-M)	M or R_1	R_2 (D16, D32, I, DS-M)	The next instruction is skipped if the test is true
	CC	M	Calc (G-M)	M or R_1	R_2 (D16, D32, I, DS-M)	The next instruction is skipped if the test is true. This instruction is normally transmitted and not executed by a CC.
JUMP	DG	Cell Bus				Usually not executed by global cells
	DL	M	Calc (G-M)	R_1	R_2 (D16, D32, I, DS-M)	The program counter is changed. Operands are used for conditional jumps.
	Ind	M	Calc (G-M)	R_1	R_2 (D16, D32, I, DS-M)	The program counter is changed. Operands are used for conditional jumps.
	CC	M	Calc (G-M)	R_1	R_2 (D16, D32, I, DS-M)	The program counter is changed. Operands are used for conditional jumps. Jumps are normally executed and not transmitted.
CC Controller Cell Instructions	CC	M	Calc	M	R_2	See Section 5. 7. 5 for a description of these instructions.
	All others					Invalid
GC Global Control	DG	Cell Bus				All operands are contained in the instruction. See Section 5.7 and Table 5-4 for a description of these instructions.
	DL	M				
	Ind	M				
	CC	M				
IO Input/Output	All					See Section 4.

Abbreviations used in table

DG	Dependent Global
DL	Dependent Local
Ind	Independent
CC	Controller Cell
M	Cell Memory Word
R_1	Register in a Cell
R_2	Register in a Cell
(X, -Y)	Modified Instruction. X's are possible alternatives, -Y is source of the alternate data or address.
G	Given address, used with instruction modification
D16	Data - 16 bits, used with instruction modification
D32	Data - 32 bits, used with instruction modification
I	Immediate Data, used with instruction modification
DS	Data List and address, used with instruction modification

Table 5-4. GC Instructions

Level Control, sent by controller cell on intercell bus	
level, G	All dependent cells at this level go to global state. Instructions follow.
level, L	All dependent cells at this level go to local control.
level, W	All dependent cells at this level go to wait state.
level, R	All dependent cells at this level reply on intercell bus with constant.
level, IND	All dependent cells at this level go to the independent state.
Format, used by all cells	
A	Given address follows the next instruction.
D16	A data word of 16 bits follows the next instruction.
D32	A data word of 32 bits follows the next instruction.
A, D16	Both data word of 16 bits and given address follow the next instruction. The address comes first.
A, D32	Same as A, D16 only the data word is 32 bits long.
I	The displacement field of the instruction is the data.
Count, DS	The number of 16 bit words given in the count field follow the given address after the instruction.
End of DS	Generated by the controller cell processor to indicate the end of the DS data.
State Control, used by independent cells	
level, DEP	The cell is made dependent, and set to the wait state. The level register is set to the value specified.
level, IND	The state is not changed, only the level register is set.

address memory; the remaining seven high-order bits are ignored. Of course, if the memory in a cell is greater than 512 words, more bits would be used. The operand is fetched from this cell's memory and placed in the specified register.

The LR instructions may be modified with a GC byte. This byte, when transmitted just before the LR instruction, modifies the address or the source of the operand. The A (address) modification forces the cell to use the given address instead of the calculated address. The D (data) modification forces the cell to load the register with the data word sent on the intercell bus. The I (immediate) modification will load the register with the displacement field of the instruction. The DS modifier is invalid.

No matter what the source of the address, the address always specifies a word in the cell's memory. Of course, the A and D modifications can not both be used with LR instructions. The controller cell may use a D modification to send the same data to all cells.

2. STR instructions store registers into memory. The address is calculated, and the contents of the specified register are placed in the addressed memory location.

A GC byte, when received just before a STR instruction, will modify the instruction. If an A modification is used, the given address is used instead of the calculated address. A data word, either 16 or 32 bits (D16 or D32), may be specified. In this case, the register contents are ignored and are not used. The data word from the cell bus is placed in the specified memory location. Thus words may be placed directly in a cells memory without changing register contents.

Both A and D may be given. In this case, the controller cell sends out both the address in which the data is to be placed, and the data to be stored. This serendipitious result is used by the controller cell to start up cells that have had their memory cleared for some reason, such as a reconfiguration.

The DS modification, when used with a store instruction, is similar to using a GC with an A and D. The difference is the instruction and address are not repeated with each data word, they are sent to the cell but once. Each time a data word is sent to the cell, the data is stored and the address is incremented. An End DS GC byte will end the sequence.

The I modification can not be used with store register (STR) instructions, because the displacement field is needed for an address.

3. OPR Instructions. The address is calculated in the normal manner, using the base (and perhaps an index register), along with the displacement in the instruction. The contents of the memory location specified by the address are obtained, and are used as operand 1. Operand 2 is always obtained from a register. The instruction specifies what operation is to be performed with the two operands. The results are always placed in a register or registers.

The following modifications are allowed.

<u>A</u> A given address may be specified, which will be used instead of the calculated address.

<u>D</u> A data word is sent on the bus, which becomes operand 1. No address is used. Depending upon the operation, the data word may be either 16 or 32 bits in length.

<u>I</u> The displacement field from the instruction sent over the cell bus becomes operand 1. No address is used.

<u>DS</u> The DS modification may be used, however, the address is not used. Because the same operation is performed with each word of data, this DS modification may not be very useful.

4. <u>RR</u> instructions operate exactly as in the independent state. No modifications are possible. Because both registers are in the same cell, no transmission of data by the cell bus is required. Only the instruction itself is sent on the cell bus.

5. <u>R</u> instructions are the same in both dependent and independent cells. No modifications are possible with register (R) instructions.

6. EXECUTE instructions can be sent from the controller cell to the global dependent cells. In this way every global cell can execute a different instruction. The address is calculated, the contents of the specified memory location are obtained and executed as an instruction. The fetched instruction may be any legal instruction for a dependent cell. The address (A) modification is allowed; the address of the memory location is sent by the controller cell. No other modifications may be used.

7. COMPare instructions are executed much differently in dependent cells than in traditional computers. In the traditional computer, a comparison is made between two values; one is located in a register. The comparison results set some flip-flops. In some computers, a separate instruction tests the flip-flops and jumps or otherwise modifies the program counter. In other machines the same instruction actually modifies the program counter. Sometimes, of course, the instruction does not modify the program counter, depending upon the results of the comparison.

The dependent cells in the global state do not use the program counter, thus another means of using the comparison results is needed. The concept adapted here is to change the level register instead.

The instruction is received from the cell bus. The address is calculated, and the specified word from the cell's memory is fetched. This word (operand 1) is compared with the contents of a register (operand 2). The results of the comparison will set a pair of flip-flops to one of 4 states. These will probably be overflow, greater, equal, less than.

Another instruction will test the state of these flip-flops and take some action. Sometimes, the same instruction may compare, set the flip-flops and take some action. The action to be taken may be one of the following. How many are mechanized will depend upon further study.

a. Continue at this level.
b. Increment level register by 1.
c. Increment level register by 2.
d. Decrement level register by 1.
e. Decrement level register by 2.

Any level register change will always discontinue the reception of instructions from the cell bus.

The conditions above are several that could be used. A compare instruction could state:

```
If   Flip-flops are 00 or 01 or 10
     THEN increment level register by 1,
     ELSE continue at this level.
```

Many other combinations are possible. How many will depend upon future software studies.

It is seen that, because the compare instruction uses data that may be different in each dependent global cell, some cells may change levels and thus discontinue receiving global instructions. In these cells, data processing will be continued at a later time when the controller cell sends out a GC for the new level.

Some compare instructions may use several words of memory, or perhaps several words from the cell bus as a DS modification. The setting of the compare flip-flops and the subsequent action is the same as an instruction that uses only two operands.

The modification possibilities have not all been explored. Some possibilities are given here.

A The given address is used instead of the calculated address.

D The data sent on the cell bus is used instead of a register operand.

I The displacement in the instruction is used instead of a register operand.

DS The words of memory starting at the given address are compared with the data words sent on the data bus. If the comparison changes the level register, the reception of data words is discontinued. If no level change is made, the flip-flops are left at their last state.

8. SKIP instructions are really test and skip. A test is made between two operands, the result of this test is always True or False. The true state will always increment the level register by 1. The reception of instructions from the cell bus will be discontinued immediately. The cell is placed in the dependent wait state. The cell will remain in this state until a GC is sent indicating instructions of the new level are being sent on the cell bus. The false state will not change the level register; reception of instructions will continue. If an address is required, it is calculated and the operand is fetched from memory. One operand is usually from a register. Some modification possibilities are:

   A̲ A given address is sent on the intercell bus.

   D̲ The data word sent on the intercell bus is used instead of the register operand.

   I̲ The displacement in the instruction is used as an operand instead of using a register operand.

   D̲S̲ The words of memory starting at the given address are tested against the words sent on the intercell bus. If any test is true, the level is changed and data reception is discontinued. If every test is false, the GC which indicates the end of the data string will be received; the level register is not changed.

9. JUMP instructions are very seldom sent to dependent active cells because they have no meaning. A Jump instruction will always be preceded by a format GC, because a JUMP, by itself, is never sent over the intercell bus. The GC will indicate a special operation is to be performed. One operation is to load the program counter with a value. The program counter is not incremented or used as long as the cell is in the global state.

10. C̲C̲ instructions are not sent over the intercell bus, but are executed only by a controller cell. Their presence indicates a malfunctioning cell.

11. G̲C̲ instructions are received by the global cells. These instructions are used by global cells to indicate the inter-cell bus data formats, and to control levels.

The format GC instructions have been described in the sections on addresses and data. These GC instructions describe what is to follow on the cell bus. Eight categories are possible (Table 5-5).

Other GC instructions may be received by a global cell. Some instructions control the levels and states. These instructions have the following formats:

GC level, G

All dependent cells at this level go to active global state. Instructions for this level normally follow.

GC level, L

All dependent cells at this level go to local control.

GC level, W

All dependent cells at this level go to the wait state.

Table 5-5. GC Formats

Cell Bus Bits 5-7	Description
0	Instructions follow (end of DS)
1	D16 (16 bit data word)
2	D32 (32 bit data word)
3	A (Given address)
4	A and D32
5	A and D16
6	I (Immediate)
7	DS (Data is being sent)

These three instructions will force all dependent cells at the given level to change state. Of course, independent cells are not changed, neither are cells that are at a different level from the level number in the instruction sent over the cell bus.

A special instruction may be used to force the cells at a level to the independent state.

GC level, IND

This instruction will set all the cells at this level to the independent state. The cell or cells will begin at the location specified by the program counter contents.

Another special instruction is the GC reply.

GC level, R

All cells at this level will respond to this GC instruction. The controller cell will now allow the dependent global cells to transmit on the cell bus. All cells which responded will now return a constant number to the controller cell. Because all cells are setting the cell bus lines to the same value, the hardware design is simple.

This instruction is used by the controller cell to determine if one or more dependent cells are at a level. The cells may switch levels dependent upon the value of the data processed by a cell. Thus some cells may or may not be at a specific level. To enable the controller cell to quickly and at low overhead determine if there are any cells at a given level, this reply instruction is included. If no cells are at this level, no cell will send back a constant to the controller cell, and the controller cell will not receive the constant reply number. The controller then may not need to send this level program. If at least one cell replies, the controller cell will note this and send out the program to process this level cells. There is no way for the controller cell to know how many cells are receiving the instructions at a level. With the reply instruction, the controller cell knows only that there is at least one cell at this level.

12. IO instructions are described in the section on input-output. Dependent global cells will usually not execute IO instructions, since the intercell bus is being used for global instructions. Dependent cells will normally be switched to local control to execute IO instructions.

5.7.2 Dependent Cells - Local Control State

The dependent cell can have its level register at a different value than the instruction and data level that is being sent over the cell bus. These dependent cells that are not active and not receiving global instructions may (1) idle or (2) execute instructions from its own memory. The first case is called the wait state, the second is local control and is discussed in this section.

The execution of instructions from the local memory will continue until one of the following events occurs:

1. An instruction puts the dependent cell in the wait state.

2. A GC instruction is received from the inter-cell bus that specifies this level.

3. A CC instruction is received on the intercell bus specifying this cell address.

In the second case, the global instructions will always be used whenever they are at the same level as the level register. The programmer is responsible to be sure that local control program is at a completed state before global instructions at this level are sent from the controller cell.

Because the instruction execution is similar to the dependent active cell, only the differences will be noted below. Note that any GC instruction modifiers must be stored in the cell's memory, preceding the modified instruction. The sequence of instructions is controlled by the program counter.

LR	The instruction is fetched from the cell's memory, the register is loaded from the memory location specified by the effective address. The calculated address is used unless a GC modifier instruction is present. The D or I modifier may be used.
STR	The instruction is fetched from the cell's memory, the register is stored in the memory location specified. The calculated address is used unless a GC modifier is present. The D or I modifier may be used.
OPR	The operation is performed between the memory location contents and the register. The GC modifier can specify an address or data word.
RR	These are executed exactly the same way in all cells. No modifications are possible.
R	The register instructions are executed the same way in all cells.
EXECUTE	Execute instructions are executed as in an independent cell. The address is calculated and the contents of the specified memory location are executed as an instruction. The fetched instruction may be any legal instruction for a dependent local control cell. A GC modifier may specify a given address.
COMPARE	instructions are executed in the same way as in a dependent global cell. The comparison is made in the same way, only the instruction and all data are obtained from this cell. The flip-flops are set in the same way.

The level register is either changed or will remain the same. If the register is changed, the cell will automatically go to the wait state. If the register is unchanged, the cell will continue to execute instructions in the local control state.

The compare may be modified as given in the global cell description.

SKIP	These instructions are executed exactly as in a global cell. The results of the test, if true, will increment the level register and force the cell into the wait state. The false state will not change the level register and thus the program will continue and fetch the next instruction. The skips may be modified, as described in the global cell description.
JUMP	These instructions will usually be executed as in an independent cell. The new value of the program counter is calculated and replaces the present program counter value. Conditional jumps are also possible. One jump will take place depending upon the comparison flip-flop setting. The jump may be modified with a GC instruction to change the level register instead of changing the program counter.

CC	These instructions are not used by a dependent local control cell since it is not a controller cell. The CC instructions are treated as no operations.
GC	These instructions can be executed by a local control cell. The GC instructions of interest are described in the section on global cells, Section 5.7.1

Only the following GC Instructions are valid in a dependent local control cell.

GC level, W

GC level, IND

All GC format control instructions

5.7.3. Dependent Cell - Wait State

This cell is not executing instructions. The program counter is not being incremented.

The dependent cell in the wait state is always examining the cell bus. When a Global Control byte is received which has the same level number as the contents of the cell's level register, the cell automatically switches to the active state and begins to receive the global instructions from the bus.

A CC control word which addresses this cell (the cell address matches the contents of the cell's ID register) will cause the cell to receive and perform the operation specified by the control word. This operation could switch the cell to another state.

5.7.4 Independent Cell

The cell operating in the independent state is described below. The independent cell operation is very similar to the traditional computer operation. These cells fetch all instructions and operands from the cell's own memory. The instruction fetched is located at the address contained in the program counter.

The independent cell cannot set its ID register. The level register, although it is not used by an independent cell, may be set to any value via a special GC instruction.

An independent cell will respond to CC commands received on the bus that specify this cell address (last name). Independent cells do not respond to level commands. The cell that is in the independent state must stay in this state until the controller cell sends a command on the cell bus with a cell address equal to the contents of the cell's ID register to change states. Thus each independent cell must be addressed individually. The independent cell concept is an important part of the distributed processor system. Other similar computer systems require all cells to be independent or all dependent.

LR instructions are all instructions that fetch an operand from a memory location and load the contents into a register. The address of the memory location is calculated by adding the base register, index register (if one is specified) and the displacement from the instruction. The low-order 9 bits are used to address memory; the remaining 7 high-order bits are ignored. A GC modifier instruction may be used, however, the address is always in the same cell.

STR instructions are the reverse of the LR, the register contents are stored in the given memory location.

OPR instructions are similar to LR, only the present contents of the register are combined with the memory location contents according to the operation code. The results of the operation are placed in the register. OPR instructions include add, subtract, multiply, divide , AND, OR, etc.

RR instructions are all instructions that use two registers, and place the results in one register. Add accumulator 1 to accumulator 2 is an example. No memory operations are required (unless the registers are stored in main memory).

R instructions are all single register operations, such as shift, complement accumulator, etc.

EXEC instruction is the traditional computer execute instruction. The specified memory location contents are treated as an instruction; this fetched instruction is executed. The independent cell can only execute instructions located within its own memory.

COMP instructions compare two values. One is located in a register, or is understood (such as zero), the other is located in another register or in the cells memory. The result of a comparison will set two flip-flops to a certain state, which will be one of 4 states. An equal comparison, for example, may set the flip-flop to 00, greater to 01, and less than to 10.

SKIP instructions are always a test and conditional skip. The value tested may be in a register or in memory. The result of a test is always true or false.

The independent cell will modify the program counter contents based upon the results of a skip test. If the test results are true, P + 2 replaces the contents of the program counter P. If the test results are false, P + 1 replaces the contents of the program counter P. In other words, the following instruction is skipped if the test results are true.

JUMP instructions are either conditional or unconditional. Additional operations may take place in addition to the jump, such as storing the program counter in an index register.

The jump is implemented in an independent cell by replacing the contents of the program counter with a new value. This new value is the location in the cell's memory where the next instruction to be executed is located. The calculation of this new value is dependent upon the type of instruction, however, in all cases, a new value replaces the old value. Conditional jump instructions make a test, usually on some register. If the test results are true, the program counter contents are replaced. If the test results are false, the program counter is handled in a normal manner, i.e., the program counter is incremented by one.

CC instructions are Controller Cell instructions. Because the independent cell is not a controller cell, the CC instructions, when fetched from memory, are always treated as no operation instructions.

GC instructions are global control instructions. All the format GC instructions may be fetched and executed by an independent cell.

The second type of GC instruction that may be executed by an independent cell is the level set instruction.

The format is:

GC   level, IND

The level number is specified by the programmer. The independent cell will set the given value into its level register and the independent cell remains an independent cell. The level register does not affect the operation of an independent cell.

Another GC instruction is the IND/DEP instruction.

The format is:

GC   level, DEP

This instruction forces a cell into the dependent local control state. The next instruction is taken from a fixed memory location One reason for requiring a special location is to decrease the consequences from a bad program accidently executing this instruction. The interrupt to a known location can verify this change of state is desired.

Input-Output instructions are executed normally, as described in Sections 4.2 of this report. In fact, the Input/Output operation is the same for all cells except those that are failed (obviously cannot perform I/O) and those in the power saving state (there are no memory words, the memory is shut off). In all other states the I/O is the same.

5.7.5 Controller Cell

The controller cell is the most difficult to describe. This cell has the characteristics of the independent cell and of a storage bank. The controller cell controls the intercell bus. The bus is used for local communication between cells and global communications. The local communication operation is described in the section on communication.

The controller cell supplies instructions and data to the dependent cells. The description here will first assume that the controller cell is transmitting instructions to the global cells, and the instructions are executed only in the dependent global cells and NOT in the controller cell. This is called the controller cell transmit mode.

The program counter in the controller cell will fetch an instruction from memory. If this instruction is a CC TA instruction, the transmit mode is entered. The CC TA instruction is described below, essentially this instruction causes thc controller cell to place the subsequent memory words on the intercell-bus. The program counter controls the fetch of instructions.

Most fetched instructions that are transmitted are NOT executed by the controller cell. The transmit mode causes non-execution (by the controller cell) of the instruction categories shown in Table 5-6. A delay between transmissions is made so the global cells will have time to execute the instructions before the next instruction is sent.

JUMP instructions are not sent out unless they are preceded by a GC modifier instruction. JUMP instructions are normally executed by the CC. The program counter is usually modified to start a new sequence of instructions as in an independent cell. Conditional jumps are executed using data from the controller cell. The address and registers used (if required) are always from the controller cell. The GC Modifier may be used if it is required to send out a JUMP instruction to the global cells.

Table 5-6. CC Transmitted Instructions

LR	Load Register
STR	Store Register
OPR	Operate on Register
RR	Register to Register
R	Register
EXEC	Execute
COMP	Compare
SKIP	Test and skip

The other controller cell mode is the execute mode. In this mode, the instructions are fetched and executed like an independent cell. All twelve instruction categories may be executed. The instructions to change the modes is described in the CC instruction description given below.

Because instructions are either executed as in an independent cell or are transmitted and not executed by the controller cell, the detailed instruction execution description will be omitted.

Controller cell (CC) instructions are always executed by the controller cell, regardless of mode. There are two groups of CC instructions: the cell address group and the controller cell mode control group.

The latter group of instructions are used to control the controller cells operation. The instructions concerned with the instruction execution and mode will be described here, those concerned with reconfiguration and interrupts will be omitted for the time being.

The mode control instructions require two bits of the instruction to specify the operation to be performed. A GC format instruction, as it exists in the controller cell memory, has two spare bits. Therefore the GC format instruction and the CC instruction may be combined to save storage in the controller cell. To simplify the explanation, the instructions will be considered separate.

The mode instructions have the following format:

CC	X
where	X is one of the following
T	Enter the transmit mode until one instruction (including any modifiers, given address, etc.) has been transmitted and then return to the previous mode.
TA	Same as T, only the transmit mode is retained until a CC with an E or EA is executed.
E	The following instruction (including any modifiers) is executed by the controller cell. Then the controller cell is to revert to the previous mode.
EA	Same as E, only the execute mode is retained until a CC instruction with a T or TA is executed.

The controller cell has many instructions to control the other cells in the group. Many of these are concerned with Input/Output and are described in Section 4 of this report.

The cell address CC instructions cause a control word to be formed and sent on the intercell bus. The control word always contains the cell address, which all cells in the group compare to the contents of its ID register. The cell whose ID register contents match the cell address will respond to the control word and perform the specified operation.

The instruction has the following format

```
CC          cell address, X
```

where

Cell address	The number to be compared to the ID register
X	is one of the following
IND	The cell is set to the independent state. The program counter is loaded from a specific location of the cells memory.
G	The cell is set to the global state.
W	The cell is set to the dependent wait state
L	The cell is set to the dependent local control state
CC	The cell is made a controller cell. The execution of this instruction automatically makes the transmitting cell an independent cell.

For the CC instructions G, W, and L instructions may follow the control word on the intercell bus. If this is done, the receiving cell will execute the transmitted instructions until a CC or a GC specifying a level is transmitted. The cell receiving the GC will go to the specified G, W, or L state, then check the level sent with the GC with the level register in the cell. Instruction modifiers may be sent with the instructions on the intercell bus.

Another CC instruction is the following:

```
CC          Cell address, level
```

The cell specified by the cell address is set to the level specified. The state is not changed.

The following GC instructions may be executed by a controller cell:

All Format instructions

No others have any meaning when executed by a controller cell. There must always be one controller cell, thus the only way a controller cell can change states is to simultaneously make another cell a controller cell. This is done with the special CC instruction.

5.8 ADDITIONAL TOPICS

The concepts described here make a powerful system with many options on how operations can be performed. Further study may show how features can be changed to improve the system. These trade-offs have not been performed. A number of alternate concepts are given in this section to show some other ideas considered.

The given address, as described above, was used without modification by the receiving cell. An alternate idea, which may aid in the software, is to have the cell always add a base register to the given address before it is used. Because each cell can have its base register set to a different value, the placement of programs and data in the cells may be made easier by using a base register with all the given addresses.

The use of a special instruction (GC) modifier was selected here because it used the least amount of time on the cell bus. Another way is to reserve a bit or so in each instruction (or memory word) to indicate its length and any special address modifications, and how much data was attached. The advantage is that each instruction carries its own length code, etc. The disadvantage is that each instruction must be made longer. The modifier makes only the modified instructions longer, but they are very much longer than they would be in the other method. The present decision was to make the unmodified instructions as short as possible, even though this made the modified ones long. The net result should be a storage saving, especially in the independent cells.

The two modes of a controller cell are only one way of selecting which controller cell words are instructions to be executed, instructions to be transmitted to global cells, given addresses, data, etc.

One method is to place extra bits on each memory word, telling what the word is. This method requires many extra memory bits in all cells.

Another method is to use two program counters. One program counter controls the fetch of instructions to be executed by the controller cell, the other program counter controls the fetch of instructions to be transmitted to the global cells. This two program counter idea allows the modes to be discarded. The resulting system is now much more elegant and powerful. However, the processor now requires more hardware. The impact is presently unknown. It is believed this idea has merit and will be studied further.

In addition, another method would be to store in the controller cell as input/output data for the intercell bus, all the instructions and data that are to be placed on the bus. This approach is very inefficient since the controller cell must identify control words by setting the control line in the bus; it has no means of distinguishing between instructions and data since they are all stored as I/O data. Of course one approach as noted above would be to have extra bits stored with each word identifying it as instructions or data with the resultant penalty of increasing the number of bits used for storage. Another method here would be to store the count of the number of words between the control words and use these counts to identify the control words. This approach requires storing a number of counts, additional control hardware and makes program modification difficult.

6. COMMUNICATION BUS OPERATION

6.1 INTRODUCTION

This section will discuss the operation of the intercell bus shown in Figure 1-1. The bus is under complete control of the cell in the group designated as the controller cell. The number of lines in the bus depends heavily on the communication rates required. Determination of this number is premature at this stage of the machine design. However, for preliminary design purposes an 8-bit parallel bus shall be assumed. This then provides for 1/2 word bytes to be transmitted over the bus (16-bit word length in the cells).

Use of the cell takes place in basically two types of modes: (a) local and (b) global. Local use is basically communication between two cells with control set up on the basis of cell address identification and not dealing with control of levels or states of cells while global use implies communication of the controller cell with one or more of the other cells with control set up on the basis of cell address identification, levels, or states for the purposes of global control and/or communication with the cells.

The bus is used for both instructions and data. Local use of the bus is basically for passing data amongst cells and amongst cells and I/O devices connected to the bus (see I/O section for a discussion of the I/O operation). Global use of the bus may be for instructions and/or data. In any case the controller cell sets up and controls all information flow over the bus. Software routines are set up in the controller cell to control the operation of the bus. There are basically two types of operations in these routines: fixed periodic and background. Fixed periodic operations on the bus are those that must take place at predetermined intervals. These may be either local or global type of communications. Examples of such operations are a cell requiring updating of navigation and guidance parameters computed in another cell every second, a set of global operations comprising a periodic program that must be computed ten times a second, etc. Background type of operations on the bus are those that take place in the absence of any fixed periodic operations; in other words operations fitted in between the fixed periodic operations.

The fixed periodic operations are generally fixed or predetermined in terms of execution time while the background are not. Therefore the routines in the controller cell can schedule or sequence the fixed periodic operations just as these types of programs are scheduled by an executive (reference 2). Background operations are scheduled between the fixed periodic operations by the routines sequentially granting access time on the bus to cells. The access time granted to the cells may be variable depending on what cell it is and loading conditions on the bus. The routines in the controller cell for handling the communication bus are part of the executive design and will be investigated in later phases of the study. This section will present the results of investigations of the operation of the bus and the type of words and formats required to mechanize operation of the bus.

A total of nine lines are used for the intercell bus. One line is used to denote control or data and is designated the control line. The remaining eight lines are used for control or data words. The control line may only be driven by the cell designated as the controller thereby prohibiting all other cells from erroneously using the control line. The lines are bidirectional so that a cell may receive or transmit over the same line; this is accomplished by the use of driver/receiver circuits at the interface with the lines in each cell (see ref. 3 for a discussion of these circuits).

The use of the remaining eight lines, in particular for control purposes will be described below. There are basically two types of control words used: Local and Global. Local words are distinguished by requiring a particular cell identification or address to be specified while global words require the specification of an identification address or certain global levels or modes. The control words are decoded by all the cells and the only appropriate cells partake in or accomplish the desired communication on the inter-cell bus.

6.2 LOCAL COMMUNICATION OPERATION

The operation of the bus in a local mode will be explained first. All words over the bus are composed of eight-bit bytes whether they are control or data words. The control words are identified as such by use of the control line; the first byte of any control word is identified by the control line set to a one (control) state. The control line will return to a zero after the first control word byte. Some control words require more than one byte, this is accomplished by use of special control word formats as will be explained below. The format of the first byte of a control word is shown below:

lines:	C	1	2	3	4	5	6	7	8
	Cont/ Data	Command			Cell Address				

Three lines are used for the command thereby providing eight possible command states and five lines for the cell address. This provides for addressing up to 32 devices or cells. The group may consist of typically 20 cells for the manned Mars mission considered, this would then leave room for 10 I/O devices to be connected to the bus (2 addresses are required for the two group switches connected to each intercell bus). It also provides for expansion so that more cells may also be added. If 32 addresses are not sufficient for addressing cells and I/O devices on the bus, it is relatively simple to provide for expansion beyond 32 by using an address extension scheme. This involves using some address, say address 32, to signify that the next byte contains more address bits. This of course has to be designed into the hardware, however it is relatively simple to implement and can provide for a high degree of expandibility both in terms of cells and I/O devices connected to the intercell bus. Of course it is possible to provide this expandibility in the cells or the I/O devices only, for example providing up to 31 cells and using address 32 for an address extension scheme in the I/O devices only, etc.

The following types of communication are required to be carried out on the intercell bus: (this list includes both local and global operations):

1. Controller cell to send words directly to a cell under controller cell's command.

2. Controller cell to receive words directly from a cell under controller cell's command.

3. Controller cell to scan bus usage requests from individual cells and establish communication between two cells based upon requests.

4. Controller cell establishing communication between two cells.

5. Controller cell issuing a command to a cell specifying some change to the internal control state.

6. Controller cell issuing global commands to one or more cells.

It should be noted that I/O devices are also included in the term cell used above. The global operations are numerous and deserve to be treated as a separate entity; therefore they will be discussed in a latter section. A description of the commands in the first byte of the control word will be given below:

Table 6-1. Communication Bus Commands

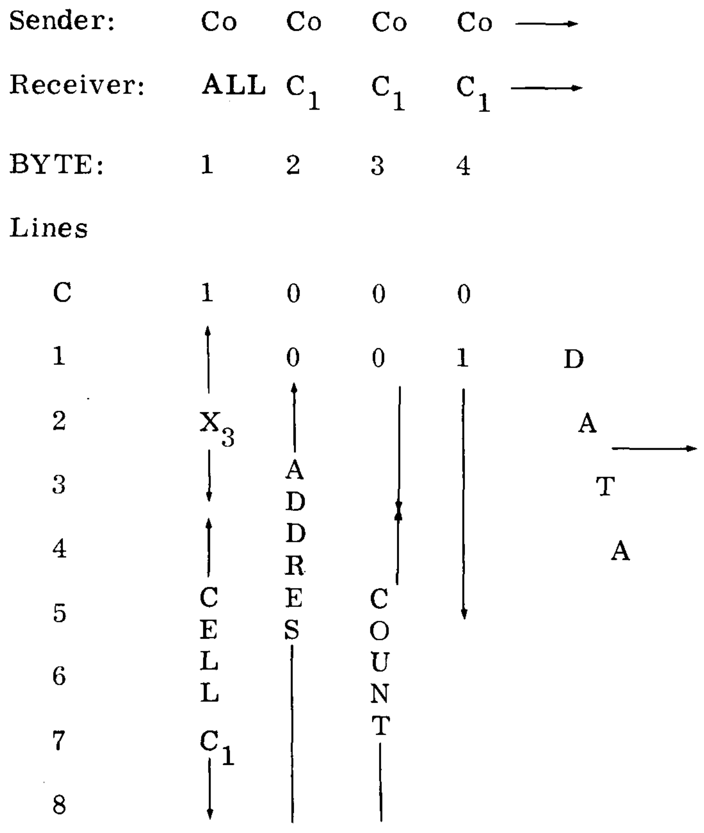
Command No.	Code	Description
X_0	000	Global mode command
X_1	001	Global mode command
X_2	010	Report communication request status
X_3	011	Input
X_4	100	Output
X_5	101	Report Status Word
X_6	110	Control Reconfiguration
X_7	111	Extended command format

Commands X_0, X_1:	These commands are used for global operations, the remaining 5 lines are not used for cell address purposes, the next section will go into detail on these commands.
Command X_2:	This command requests a response from a given cell as to the status of its requests for use of the communication bus.
Command X_3:	This command tells the cell to input the next set of data words on the bus. (A set of data words is defined as the words on the bus in between command words)

Command X_4:	This command tells a cell to output a set of data words on the bus.
Command X_5:	This command requests a cell to send to the controller a status word representing certain control states in the cell.
Command X_6:	This command forces a cell to perform some change to the internal control state (e.g., turn on/off, etc.).
Command X_7:	This command uses an extended format. It requires the second control word byte to identify what the command consists of; this then provides for more than the eight basic commands listed here. This command will also be used to change cells from local control modes to global control modes as will be explained in the next section.

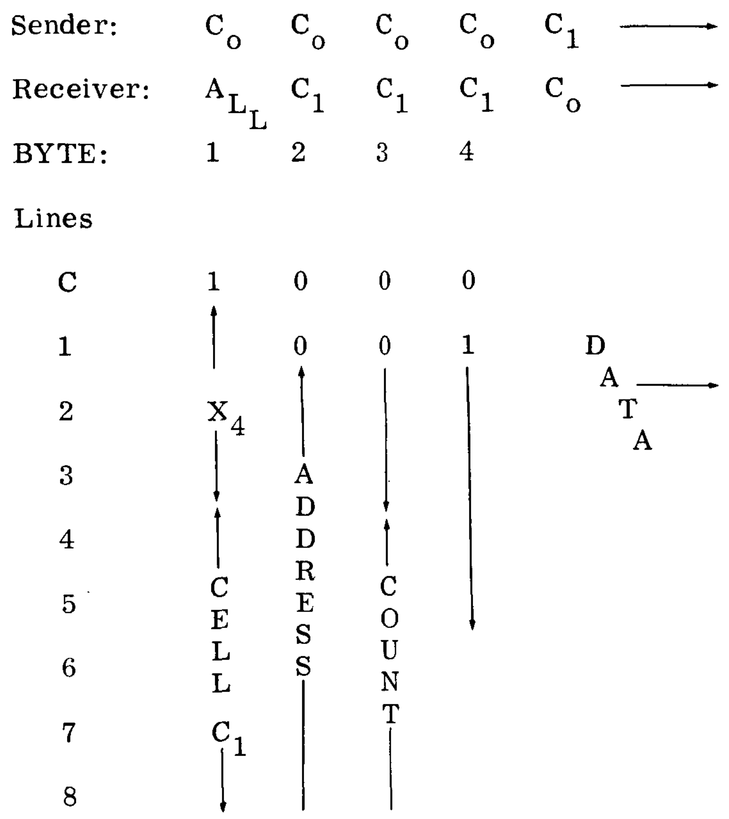
The mechanization of the required communication operations on the bus will now be discussed:

1. Controller to send words to a cell:



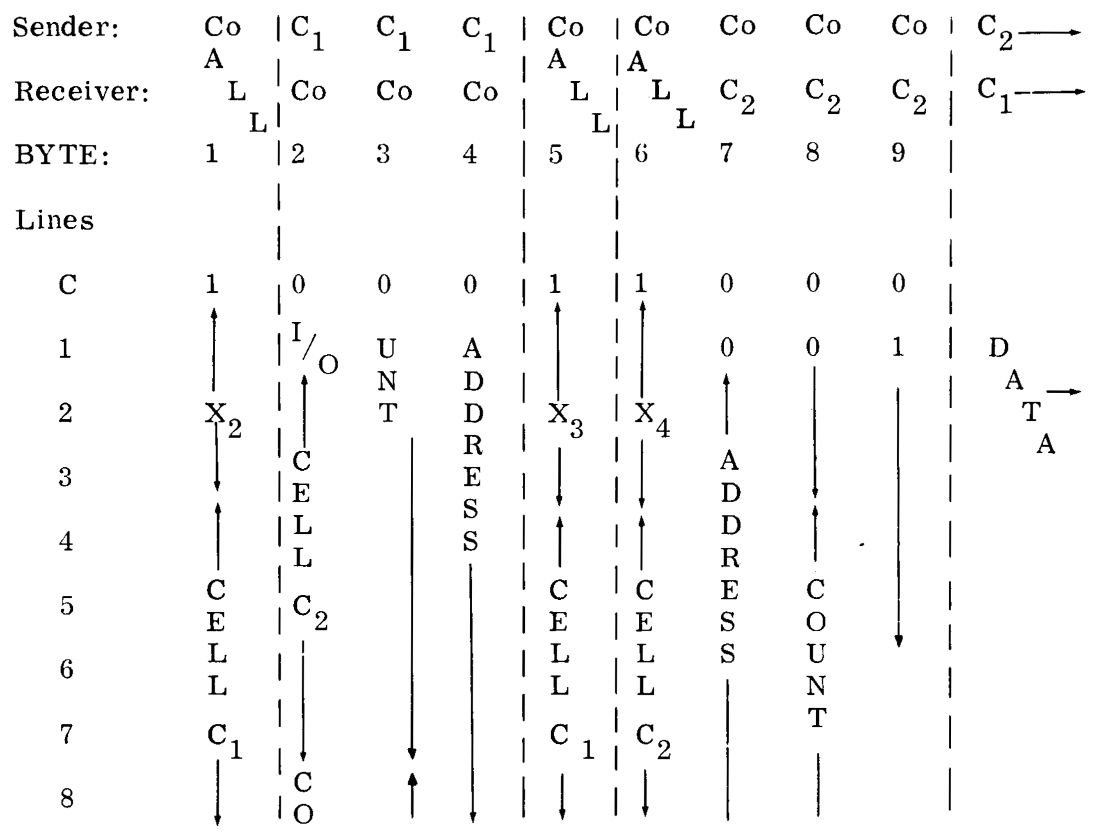
The first byte of the control word is identified by a one on the control line. As mentioned previously the first byte contains the command, input, and the address of the cell, C_1. A 9-bit address is sent in the second byte and part of the 3rd byte to identify the first location in the cell to be input to. The number of words to be input to the cell are specified by the count which is sent in the remaining part of the third byte and part of the fourth byte (if needed). The use of only three bytes in the control word provides for a count of up to 32 words. If a count of more than 32 is to be input then a fourth byte is sent to the cell. The cell determines when the control word is complete by the 0 to 1 transition on line number 1 as shown above; this scheme provides for the capability of a variable length control word format and saves the transmission of one byte when less than 32 words are to be input. The following definitions will be used to identify the cells: C_o, the controller cell, C_1 and C_2, the cells used in the communication process.

2. Controller to request words from a cell


Sender: C_o C_o C_o C_o C_1
Receiver: A_{L_L} C_1 C_1 C_1 C_o
BYTE: 1 2 3 4
Lines
C 1 0 0 0
1 0 0 1
2 X_4
3
4
5
6
7
8
CELL C_1
ADDRESS
COUNT
DATA

The same discussion as for 1 above applies here except that the control word applies to outputting data from cell C_1 now.

3. Controller to scan a cell for communication requests - Cell, C_1, to request words from cell C_2.

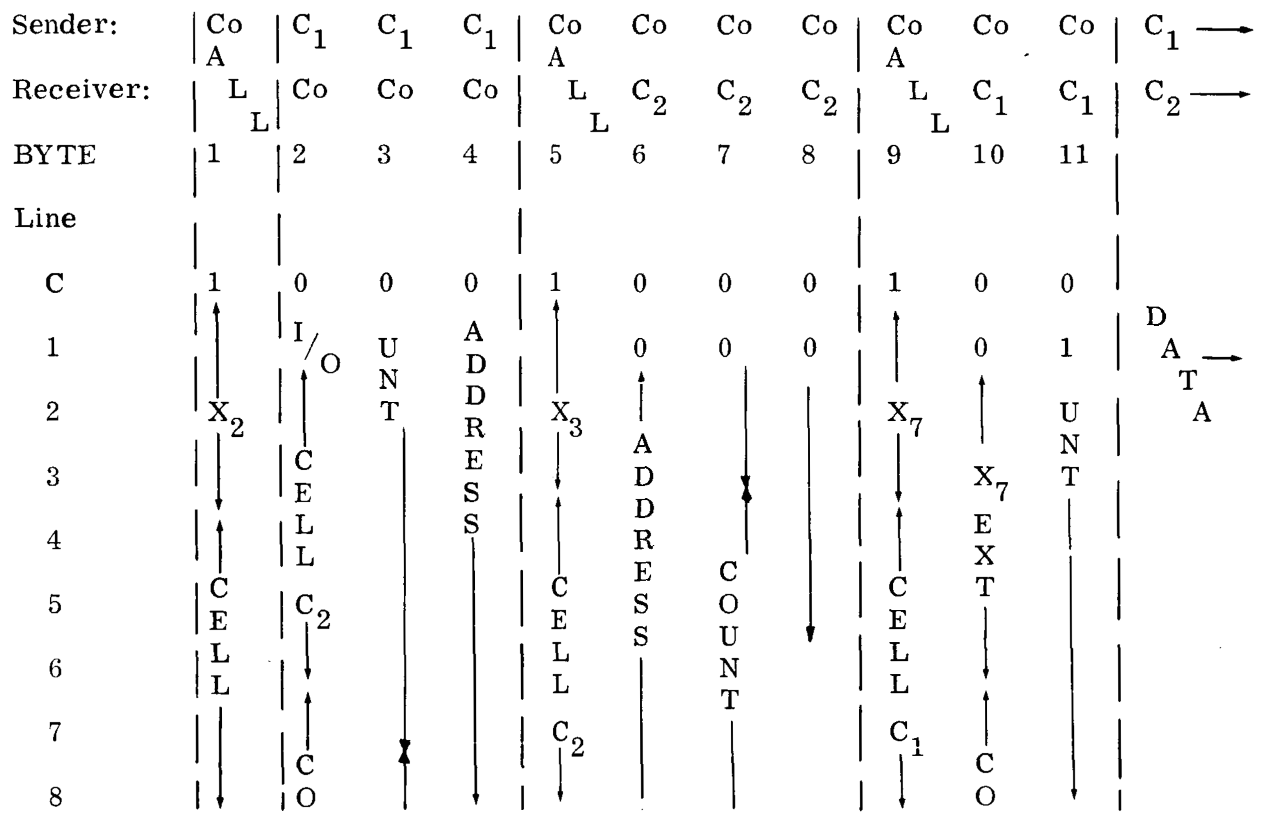


The controller cell outputs the first byte of the control word sequence, X2, which asks a particular cell, C_1, if it has a request for service on the bus. Cell, C_1, responds by outputting a response word as shown above. In this particular case the first byte of the response identifies the desired operation (input for this case), the Cell, C_2, communications is desired with, and part of the word count, the address and number of words are completed in the remaining two bytes. The address in this case specifies the location in cell C_2 from which the words are desired. If a cell has no request the response will consist of two bytes all zeroes. It should be noted that cell C_2 could also be the controller cell with whom a request is made for communications with.

The controller cell accepts the response from the cell and examines the request. It will determine whether the full word count requested can be accomodated on the communication bus. The controller cell then outputs the fifth byte of the control word which is an input command to the cell, C_1, that requested the words. Next the controller cell outputs the remaining control word bytes telling cell C_2 to output a certain number of words (may be reduced below that requested by C_1) starting at a location specified by the address. The same comments apply as before with regards to varying the length of the X_4 command should less than 32 words be desired to be communicated.

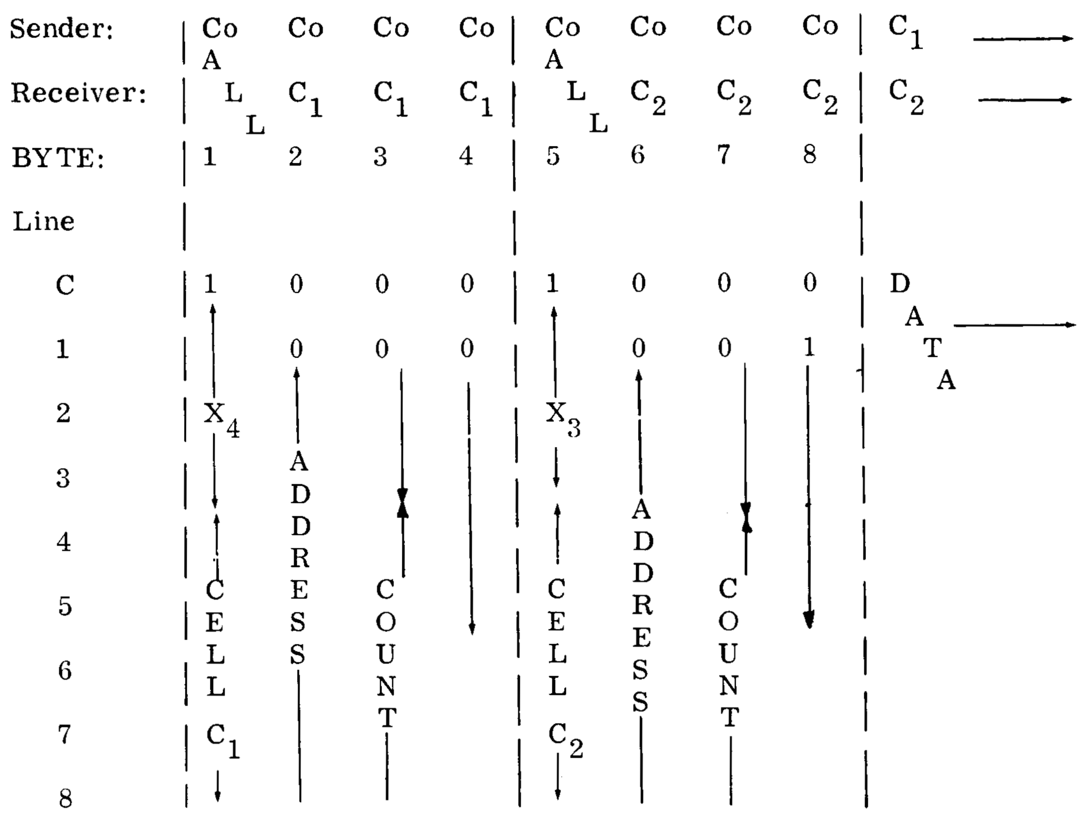
It should be noted that the X_3, input, command to cell C_1 given by byte 5 is not executed until byte 9 has been sent. This is due to the fact that the X_3 command is a variable length command and requires a 0 to 1 transition of line 1 after transmission of the first byte of the control word comprising this command to signify the complete transmission of the command. This logical function is utilized here so that cell C_1 is told to input, however it will not pick up words on the bus until byte nine has been transmitted (this 0-1 transition will enable the receiver circuits in cell C_1 and simultaneously enables the driver circuits in cell C_2). It should also be noted that the byte 6 command is actually not examined by cell C_1 since it has a command which is in the process of being set up (normally a 1 on the control line forces all cells to examine the command to determine if it is for them). Also note that the number of words that will be received by cell C_1 may be different (less) from that requested by it; the cell keeps track of the difference if any between the number of words requested and actually received. If any difference exists it will take the appropriate action on its next request to the controller cell.

4. Controller to scan a cell for communication requests - Cell, C_1, to request to send words to cell C_2


Sender: CoA | C1 C1 C1 | CoA Co Co Co | CoA Co Co | C1
Receiver: LL | Co Co Co | LL C2 C2 C2 | LL C1 C1 | C2
BYTE 1 | 2 3 4 | 5 6 7 8 | 9 10 11
Line
C 1 0 0 0 1 0 0 0 1 0 0
1 0 0 0 0 1
2
3
4
5
6
7
8
X2 CELL
I/O CELL C2 CO
UNT
ADDRESS
X3 CELL C2
0 ADDRESS
0 COUNT
X7 CELL C1
0 X7 EXT CO
1 UNT
DATA

The mechanization of this operation is very similar to that described above for the input request by a cell. In fact the first four bytes are identical except that an output is indicated in the second byte. The cell, C_2, to whom words are to be sent is told to input by bytes 5 through 8; note that byte 8 does not have line 1 in a 1 state. This prevents cell C_2 from picking up the next three bytes as data words. The requesting cell, C_1, is told to output by means of an X_7 extension command (output - word count), this is used since all the controller can tell cell C_1 is how many words it may output since it does not know from address they will come from (if it did it could use an X_4 command). The X_7 command is a variable length command and therefore uses a 0 to 1 transition on line 1 after the first byte to signify its completion; note that this transition is also used to complete the X_3 command given previously to cell C_2.

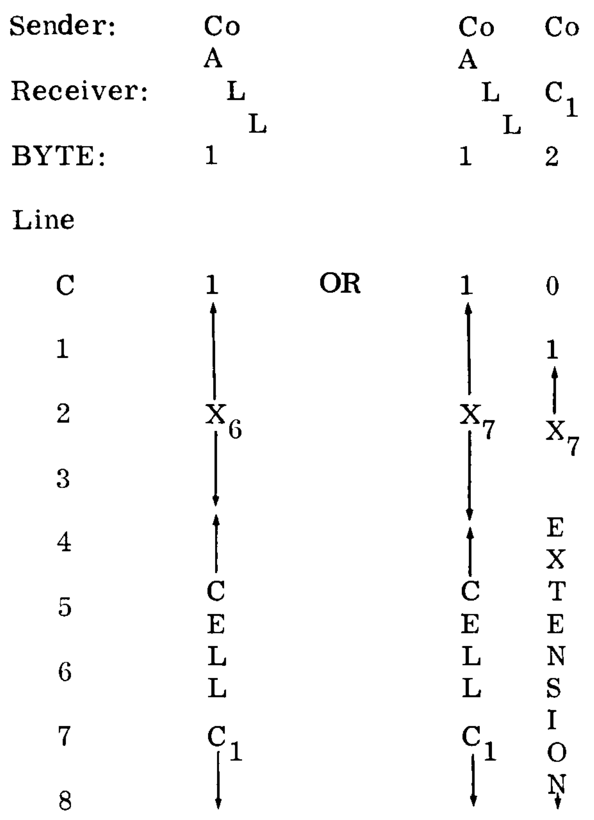
5. Controller Cell to tell one cell, C_1, to output to another cell, C_2.



This operation requires an output command as described previously to be given to the cell C_1 and an input command as described previously to cell C_2. The only difference is that a 0-1 transition on line 1 is inhibited in byte 4 so that cell, C_1, may not start sending data until byte 8 has been transmitted.

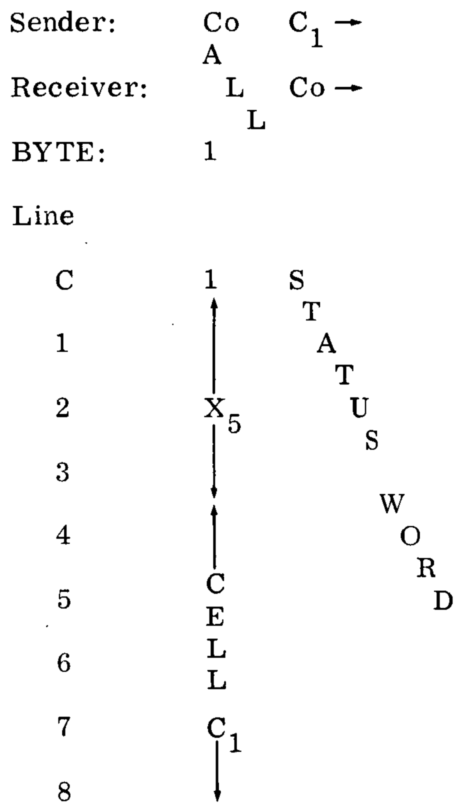
6. Controller Cell to tell one cell, C_1, to input from another cell, C_2. (Same as above.)

7. Controller Cell to command a cell, C_1, to reconfigure some control state.



This command is simply a one byte command if the control change is represented by X_6, two bytes are necessary to identify other changes by using X_7.

8. Controller to Command a cell, C_1, to report its status word.



This command forces cell, C_1, to output a fixed location status word to the controller. The format of the status word will be determined in later phases of the study.

It should be noted that there are a number of alternatives in deriving the formats presented above. For example, it is possible to only use one byte for the X_3, input command, since the first word transmitted could contain address and word count information. However, this provides no transmission time saving on the bus and contaminates the data being sent and received with control information. It is also possible to not use the variable command scheme requiring the 0-1 transition on line 1 for certain commands (X_3, X_4 and X_7). It would thereby provide for up to 128 words with three bytes for the X_3 and X_4 commands. To go beyond 128 words, one could use a word count of 128 to signify that the next byte is to be used as an extended word count, this could result in some savings on the bus. However it would not be possible to use the 0-1 transition logically to delay the start of certain X_3 and X_4 commands as was explained above. To accomplish this would require additional logical circuitry and/or transmission of additional bytes so there may not really be any savings in eliminating this logical function.

6.3 GLOBAL COMMUNICATION OPERATION

This section will present a description of global control of the inter-cell bus. The same format is used as presented in the preceeding section (6.2). It was pointed out there that commands X_0, X_1, and X_7 are used for global control. Commands X_0 and X_1 do not specify a 5 bit cell address identification but use the bits for control purposes; these commands will be one 8 bit byte long. Command X_7 will utilize a 5 bit cell address identification, as mentioned previously it uses command extension and is variable in length to offer the possibility of many control commands within X_7.

6.3.1 Format GC Instructions

One byte is used here and the format is shown below:

lines:	C	1 2 3	4 5	6 7 8
	1	0 0 0	Y	L

Y	=	11	indicates format type GC
L	=	000	indicates end of DS data words
		001	indicates A - given address
		010	indicates D16 - 16 bit data word
		011	indicates A and D16
		100	indicates D32 - 32 bit data word
		101	indicates A and D32
		110	indicates I - Immediate data
		111	indicates DS - Data is being sent (A is also sent here)

6.3.2 Level Control GC Instructions (Class 1)

The same one-byte formatas given above is used here.

L is always the level number to be compared to the contents of the level register in the dependent cell.

Y = 00 G - indicates all cells at this level go to global state, instructions for this level follow

01 L - all cells at this level go to local control state

10 W - all cells at this level go to the wait state

6.3.3 Level Control GC Instructions (Class 2)

One byte is used here and the format is shown below:

lines:	C	1 2 3	4 5	6 7 8
	1	0 0 1	Y	L

L is always the level number

Y = 01 R - all cells at this level reply with a constant, to be sent on the inter cell bus.

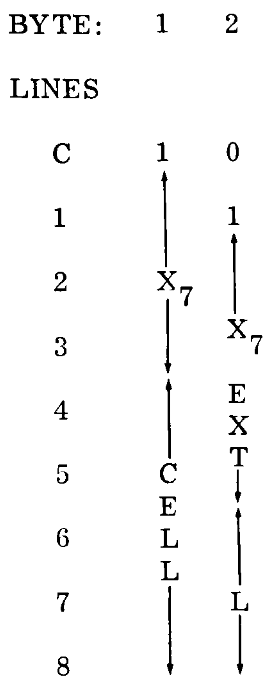
10 IND - all cells at this level go to the independent state.

00 Spare

11 Spare

6.3.4 Individual Cell GC Instructions

Recall that the X_7 extended format command is used here, the format is shown below:



The three variable fields have the following meanings:

CELL	The cell address that is compared to the cells ID register.
L	The level number that is to be loaded into the cell's level register.
X_7 EXT	The operation to be performed by the receiving cell.

When X_7 EXT	=	1000	IND	Go to independent state
		1001	G	Go to dependent global state
		1010	W	Go to dependent wait state
		1011	L	Go to dependent local control state
		1100	CC	Go to the controller cell state.

The remaining X_7 EXT codes are used for other operations, one of which was given in Section 6.2.

The use of the instructions presented in this section was given in Section 5 and reference should be made to clarify their use in global operation of the bus.

7. MACRO INSTRUCTIONS

The DAMP system is basically an array of cells consisting primarily of storage with a small amount of each cell devoted to a processor (arithmetic and logical). An important consideration is what can be added to the processor section that can result in requiring less storage and a net reduction in total hardware required in the cell. Macro instructions (MACROS) were one such feature considered and this section will give a brief discussion of Macros in general and several specific types investigated.

A considerable amount of additional effort in this area would be necessary in order to choose a set of Macros to add to the set of common instructions (add, sub, transfer, etc.). In particular the amount of hardware necessary to implement a Macro should be traded-off against the amount of storage necessary to implement it as a subroutine using common instructions. In addition, it should be determined how much each macro would be used, since including it in the instruction repetroire requires including it in all cells. This would clearly only be worthwhile if the Macro was used often and it required a relatively small amount of hardware compared to the amount needed for common instruction storage necessary to implement the same operation. The above trade-off is biased against including most macros that may be suggested. In fact the situation is even worse when it is considered that many functions that are candidates for macros, such as sine or cosine, could be implemented as a common subroutine in a single cell devoted to receiving parameters from other cells and sending back sines, cosines, etc. as required. As a result storage for certain routines may only be required in a small number of cells.

7.1 CORDIC ALGORITHM

The Cordic Algorithm is adequately described in the literature as a useful means of generating sines, cosines, and other trigonometric or hyperbolic functions (see ref. 4 and 5). Some consideration was given to including the hardware necessary for an efficient use of the algorithm in the processor sections of the cell. The algorithm could of course be implemented by programming with a normal instruction set; however this would require more instructions than the typical series solution implemented. (The series solution for sine and cosine simultaneously requires on the order of 30 instructions depending on the machine). As a result consideration was given to making the necessary additions to the general purpose hardware in the cell. (The cordic hardware does not provide sufficient flexibility to replace the GP hardware; however it may enable some instructions to be deleted from the normal instruction set.)

The hardware used to implement the algorithm typically involves three registers capable of being shifted, two adders connected to two of the registers so that cross addition of two of the three registers can occur, a third adder for the third register, gating hardware to enable a variable pick off from two of the three registers (this enables 2^{-j}, $j = 0, 1, 2 \ldots n$, times the contents of a register to be picked up for cross addition), and control circuitry. In addition to the above, for an n bit word, n angle constants must be stored either directly in fixed hardware or in the memory. The above registers can be simply made available from the normal processor registers (accumulators); however it would be necessary to add additional connections, adders, gating, control circuitry, and possibly the n constants. (If the constants are not in fixed hardware they must be accessed from memory via stored instructions or by control circuitry.) In any case, implementation of the cordic algorithm, including the stored constants, would require a few thousand FET's in addition to the 5000 used for the general purpose hardware. Even with this hardware the algorithm still requires a number of instructions at least for initialization and storing the result. The hardware implementation of the algorithm would offer increased computation speeds for trigonometric functions, but this is not what is needed in the distributed processor system.

An accurate comparison can not be made of the above hardware that would be required in every cell with the number of instruction locations in the whole machine that would be required to execute the same functions. However, the applications for which the cordic algorithm would be useful, navigation problems involving sines, cosines, coordinate transformations etc., represent only a small percentage of the requirements for the space missions under consideration. As a result trigonometric routines would only be required in a small percentage of the cells. In fact use of separate cells for subroutine storage and execution as mentioned in the introduction would reduce to an even smaller number the percentage of cells storing trigonometric routines. From the above discussion it can then be realized that increasing the complexity of each processor by the addition of cordic hardware would bring a small return in additional available memory. As a result it is considered to be not worthwhile to implement this algorithm. (A relatively large increase in processor complexity of course makes the processor more difficult to fabricate and can result in lowered wafer yields, as a result complex macros should not be included unless they save a good amount of storage.)

7.2 DDA

The cell could be made into a DDA-GP structure. The DDA portion of the machine could be used for generation of trigonometric and hyperbolic functions as with the cordic hardware; however this DDA-GP organization would not be worthwhile for the same reasons as for the cordic algorithm. Quite a large number of FET's would be needed since at least two complete DDA's would be required including programming flexibility for the interconnections. In addition, the DDA implementation would require a number of memory words for initialization and storing the result.

7.3 GENERAL MACRO SET

Macros are being considered for the DAMP System primarily from the standpoint of saving storage. As a result, the present investigation of macros is pointed toward those that would replace a number of common instructions (decrease the number of bits associated with instructions so that less instructions can be used) and that would be used in a reasonably large number of computations. Actually, limited usage of macros would be acceptable if they did not require very much additional hardware in the processor.

The following paragraphs discuss a few basic types of macros in order to point out the type of macro that is the most fruitful to investigate for storage savings. The first type presented are basic instructions (add, etc.) that operate on non-ordered lists of data (i.e., each data word must be individually addressed). These macros save some bits, but it is felt they will not be used very often in the applications of interest here. A second type of macro would again carry out basic instructions but on ordered lists of data. These macros are shown to be of relatively small value because they only replace loops that generally contain a very small number of instructions. The third class of macros are characterized by complex instructions on any type of data. These macros are shown to offer the best possibilities for storage savings; as a result, investigations of macros should emphasize this latter type.

It should be emphasized that when trying to save memory, the number of bits used by a macro instruction is of importance. There are a number of types of macros that can be investigated for memory bit savings. One example is multi-operand macros that individually address a number of operands (operation on non-ordered data that must be addressed from random locations) and carry out a basic logical or arithmetic operation on all of them. These macros require sufficient bits to address each operand; as a result the only memory savings is due to the saving of the op code bits that would be necessary to individually access and combine the operands. This saving could be large if enough operands could be combined at once. However, for macros that would be used sufficiently it seems that only a very few operands could be combined at once as described above and as a result the bit savings would be typically small.

Many operands with a single basic arithmetic or logical operation are typically carried out with data that is ordered into some type of a list in memory so that inherent operand addressing is possible. An example of this type of addressing is the processing of a list of information with an instruction loop that uses index registers to hold and update the operand addresses. (Note that the list does not have to be simply sequential. It could use every other memory location, etc.) The inclusion of a repeat mode in a processor enables loops, as described above, that contain a single instruction to be executed very quickly since the count of the times through the loop and the termination of the loop are automatically handled. However the use of a repeat mode or of a macro to initialize the appropriate index registers and execute an operation on a list of ordered data can save very little storage since the loops used to carry out the same operation require only a few initialization words for the appropriate index registers plus the basic loop (one operation plus index handling instructions). As a result a macro to carry out a basic operation on an ordered list would save very few bits (only the op code bits for the index initialization and handling instructions). The inclusion of such macros or even a repeat mode is then not worthwhile in the DAMP System unless an increase in the speed of execution is needed. Another type of macro that could use basic arithmetic or logical instructions and forms of inherent operand addressing to save storage bits involves using a push down stack in the processor. Data could be processed and placed in the stack, and when appropriate a macro could be issued that executes a basic arithmetic or logical operation on the top members of the stack. The number of words to be combined would be the only parameter required in addition to the op code (no addresses). This macro would then save the loop initializations and index handling that would be necessary if the same instruction was executed on the list of data by a loop. The stack may also find other uses for the programs, however, it is not clear at this time that such a stack would find any real usefulness.

The class of macros characterized by a single instruction that replaces a quantity of basic instructions offers a good opportunity for memory bit savings; however, macros of this type that will be used fairly often and do not take an unreasonably large amount of hardware are difficult to find. Useful macros of this type could operate on very few operands or could use some form of inherent operand addressing to operate on lists of operands. In fact use of the cordic algorithm to generate trigonometric functions could be considered an example of such a complex macro. This macro could call out sine, for example, and then give the angle to a hardware unit set up to return the answer; however, this macro was shown to be impractical from a hardware standpoint since it provided high speed sine generation but saved very little storage and was not used very often. Exactly the same arguments eliminate consideration of special function generators using diode arrays, for example. One possible useful instruction of the complex macro type is the vector dot product. It would address the first element in each vector and place the addresses in index registers. The instruction would also specify the number of elements in the vector and place this value in a third index register. The elements of the vectors would be stored in the memory in sequence following the first element. The instruction would then address all elements by simply incrementing the index registers, multiplying corresponding components, and adding to the previous sum (stored in the second upper accumulator and the lower accumulator) until the third index register reaches zero and the operation is terminated. This macro would then save the bits necessary to specify the initializations and the sequence of operations within the loop that could be used for the calculation. The main usefulness of this instruction would be in executing matrix multiplies, however it may also find use as a substitute for a sum of products multiply. The difference in this latter case is that two memory locations would be multiplied instead of one of the upper accumulators and a memory location. This is acceptable except that more bits are required than for specification of a simple sum of products multiply; therefore the vector dot product instruction would be an improvement in this case only if the use of the sum of products instruction generally requires the accumulator to be loaded first. (This is probably not generally the case.) In order to get a good evaluation of the value of this macro its use in a matrix multiply needs to be compared to a loop implementation of a matrix multiply. The amount of usage of the instruction should also be evaluated. Other macros of the above complex type need to be investigated. Two possibilities are a full matrix multiply and complex operations on a stack.

8. PROCESSOR DESIGN

8.1 PROCESSOR FEATURES

This section will describe the general features of the processor section of each cell (actually the I/O portion of each cell is also included in the processor section). In particular, the word length, accumulators, index-bank registers, and the instruction word format will be discussed. Since the requirements to be designed to are not very firm at this point in time, some of the alternatives discussed cannot be explicitly chosen as optimum. In addition, certain types of features such as those associated with the global mode operations and communication bus are presently under investigation and are therefore not included in the processor design. Therefore, even though this section is not complete, it is included to give some perspective as to the description of a cell.

8.1.1 Accumulators and Index Registers

This sub-section will discuss the use of multiple accumulators and multiple index registers in terms of their ability to save storage. Since it is not necessary for the processor to operate at high speed, as many of the processor registers as possible will be stored in the memory. This enables many registers to be used for the processor at a small increase in system complexity or lowering of wafer yields. This is the case since registers in the memory can be easily fabricated with high yields by using discretionary wiring or similar techniques. On the other hand, the registers constructed in the processor are generally part of lower yield complex logic. (If this differences in ease of fabrication is eliminated in the future the multiple accumulators and index registers can be included in the processor section of the chip. The instruction execution time would then be decreased.) Because of the above points there will be just one accumulator in the processor section of a cell and any additional accumulators will be accessed from a specified area of the memory. (The chosen processor uses four accumulators.) The index-bank registers will also be contained in a fixed area of memory.

Accumulators

The use of more than one accumulator can save a significant amount of execution speed and, of more importance for this application, storage. The storage savings comes about since intermediate results do not have to be stored in the memory or in hot storage. As intermediate results are obtained they are simply left in the accumulator in use and another accumulator is brought into use. In this way it is not necessary to store the first accumulator while further operations are carried out before the intermediate value is again needed. The accumulator to be used in any operation is simply specified in the op-code. As a result when the instruction is to be executed the proper accumulator is pulled out of the memory and exchanged with the accumulator in the hardware location or if the hardware accumulator is the one specified no exchange is necessary. This process clearly takes a longer time than if the accumulators were in the processor hardware itself; however, for the processor of interest the main interest is in saving storage and as a result the processor hardware that would be devoted to multiple accumulator registers can be eliminated or used in other fashions, e.g., for complex macro control, etc.

The usefulness for memory savings of a second, third, or more accumulators for intermediate storage must be evaluated for any given application. However, an evaluation that was carried out (Ref. 6) showed that for guidance and navigation in an avionics system addition of a second accumulator reduces the instruction count by as much as 8 percent and the inclusion of six or more accumulators bring this reduction to as much as 12 percent. How much the saving is for scientific experiments, telecommunications, etc. has not been determined, but it is clear that the use of at least 2 accumulators is a very valuable asset for saving storage. Since these accumulators are in memory, their only hardware cost is additional control circuitry. It can be seen that the majority of the advantage of multiple accumulators is accrued with the addition of the second accumulator, as a result each processor cell will have at least two accumulators. The inclusion of additional accumulators depends on both the availability of instruction bits to specify to which accumulator a particular op-code is being applied and the relative usefulness of additional accumulators versus additional index registers. These points are made clearer by the discussion on each possible word size given later.

Indexing

Full word length index and bank registers are used. Therefore, there is no real distinction between the two since they both accomplish address generation and address modification in the same manner. They will be referred to as index, bank, or index/bank registers throughout this report.

Indexing in each processor of the cells will be carried out by memory index registers. For an instruction that requires banking or indexing, the proper index/bank register or registers are accessed from the memory, loaded into the memory buffer register, and added to the memory address register (the memory address register holds the instruction displacement obtained from the initial instruction word). From this it can be seen that an instruction that needs to be banked and indexed before picking up the operand would require four memory cycles, including the memory cycle to pick up the instruction itself (accumulator may be in memory).

The advantage of indexing in terms of memory saving has been discussed in many places. Two such discussions are given in References 6 and 2. From these references and from an investigation of the requirements, it can be seen that the inclusion of at least three index registers along with one or more bank registers will provide significant storage savings (20 percent or more); however, the addition of more index registers than three provide significantly less storage savings. As a result, at least three index registers will be included in the processor of each cell, the use of additional index registers depends on the availability of instruction word bits and on a comparison of the value of these additional registers to the value of additional accumulators. The index registers could also be used for temporary storage; however, use of the index registers in this manner would provide no memory saving unless register to register instructions were included (instructions carrying out basic operations such as add from one index register to another index register). The reason for this is that the index register used as temporary storage cannot be addressed directly by bits in the op code if an operation is to be carried out between the temporary storage and a memory operand. This is clear since there are index register bits in the instruction word that must specify the indexing of the address for the memory operand. Therefore, only operations that address two index registers used as temporary storage and then add or subtract them, etc., could provide an instruction saving over the use of memory addressing for intermediate results. On the other hand note that if accumulators are used for temporary storage, they may easily be addressed by additional bits in the op-code portion of the instruction word since these registers will not be used for indexing a memory address. Investigation of the usefulness of register to register operations versus the usefulness of providing accumulators that may be used for temporary storage has shown that the register to register operations find very little usage in comparison to the accumulators. As a result using accumulators instead of index registers for temporary storage provides a much greater storage savings. (Reference 2 discussed the use of hardware index registers for temporary storage in order to provide increased execution speed by using these registers for temporary storage. Note that this increased speed cannot be obtained for the processor specified above since the temporary storage or index registers are located in the memory and must be accessed with a memory cycle in a similar fashion to any other operand held in the memory.)

Indirect Addressing

Investigations of the usefulness of indirect addressing have been carried out and are discussed in reference 2. It was found that indirect addressing in a machine with a number of index registers had a limited usefulness; however, when it was used, it provided some storage savings. (The primary use was for sub-routine linkage.) As a result of this limited usage it is not recommended that an indirect bit be added to the instruction word, but that instead where applicable, certain instructions may use only indirect addressing or may have the facility to use indirect addressing if desired.

8.1.2 Word Size

The desire to save storage (to use the least number of bits in the memory possible) gives a reason for considering small word sizes. If a small instruction word can include enough features, it may provide enough flexibility such that it would require only a slight increase in the number of words for instructions in the memory, over that for larger instruction word. This may then result in smaller number of bits in each cell's memory for instructions. Larger instruction words are generally used to offer more flexibility and increased processing speed. The increased processing speed is not required here, but the amount of storage saved by increased instruction word flexibility must be investigated. It is also necessary to determine the amount of extra data words that would be necessary with a small word size. A small word would require increased double precision and possibly some triple precision operations and would result in smaller byte sizes for storage of multiple bytes per word.

Twelve-Bit Word

A 12-bit instruction and data word was first investigated to see if it would offer enough flexibility for a savings in the number of bits in the memory over a larger word. The chosen 12-bit instruction word is shown in Figure 8-1. Five op-code bits should be sufficient to offer a reasonably large and flexible instruction repertoire including the ability to use 2 accumulators. More than 32 instructions can be made available by the use of op-code extension on instructions that do not require a memory address. Two uses of the tag bits I/B, are shown in Figure 8-2. Figure 8-2a would have a bank register, B, contained in the memory added to the address bits in the

bits:	1 - 5	6 7	8 - 12
	op code ^{5}	I/B ^{2}	Address Displacement ^{5}

I/B = index/banking bits

Figure 8-1. 12-bit Instruction Word

0 0	B	0 0	T_1
0 1	$B + T_1$	0 1	T_2
1 0	$B + T_2$	1 0	T_3
1 1	$B + T_3$	1 1	T_4
	a.		b.

Figure 8-2. Use of Two I/B Bits

instruction word for every memory access. In addition, one of three memory index registers can also be added to the address. Figure 8-2b proposes using the index/banking bits to specify one of four index/bank registers. Since this scheme does not enable multiple indexing to be carried out, it will require a few more instructions for execution of loops. However, it does not have the disadvantage of the scheme specified in 8-2a, of requiring that the index register contents be changed any time the bank register contents are changed. None the less, the scheme in Figure 8-2a seems to be somewhat more flexible and would be chosen for a 12-bit word.

The last part of the instruction word provides 5 bits for specification of an address within a bank. Clearly this is only a 32 word bank; as a result it is probably too short to hold the majority of the programs and/or their associated data. This means that a reasonably large number of load bank commands would have to be inserted into the instruction stream both for jumping to separate parts of a program that is located in a number of banks and for picking up data for separate banks. A complete investigation of the programs is necessary to determine the percent increase in storage due to these load bank commands; however, an investigation (reference 7) showed that for navigation and guidance programs a 32 word bank could cause substantial increases in the amount of memory required. This short bank is especially inefficient in a 12 bit word where the index/banking scheme is relatively inflexible.

There are a number of additional problems with a 12-bit word. One of these would be the inability of a memory word to contain the complete address for any word in a group. A group may typically contain on the order of 20 cells of 512 words per cell. This would amount to at least 10,000 words per group, whereas a 12-bit word can only address 4,000 words. It is presently felt that it will be reasonably common for one cell to address a memory location in any other cell in its group; as a result, a 12-bit word would require two locations to hold this address and the second location would only need two of the 12 bits for additional addressing. This could amount to a substantial inefficiency.

In addition, the use of a 12-bit word would certainly require triple precision operations to be carried out in the navigation and guidance routines. This could cause an inefficiency of bits in the data word and would also require the addition of triple precision software. It also appears from the requirements that the half word size or byte size of 6 bits would be somewhat small for the needs of many of the scientific experiments and other operations that will use byte manipulations. In particular, a seven to eight bit byte would probably be necessary to offer sufficient flexibility. A precise answer to the size of the byte required for efficient utilization of data storage is difficult to provide; however, 7 or 8 bits would certainly offer more flexibility to meet the requirements for byte manipulation when they are explicitly specified.

It should be noted when discussing byte manipulation that the number of bytes per word should be a power of two for the most ease of operating on bytes. (The most flexible byte manipulation is when the number of bits in the word is a power of two.) If this is the case bytes can be manipulated and obtained from data words by simple shifts of the addresses of the bytes. In addition, indexing with respect to words or with respect to bytes can be done simply by adding or by shifting and adding the specified address to an index register.

The addition of a number of half word or byte instructions to the instruction repertoire, such that bytes are directly accessed and operated on and then replaced in memory without affecting the remainder of a memory word, may save a considerable amount of storage (e.g., in the scientific experiments in a spaceborne application). The addresses of these instructions could handle the additional length required due to byte specification since they would be relative to an initial address of a word in a list. (The initial address would be held in an index register.) Therefore, since a considerable amount of byte manipulation is expected in at least the scientific experiments, only word lengths that are a power of two and a multiple of some useful byte length, such as 6, 7 or 8 bits, will be considered, i.e., 12, 14, and 16 bit word lengths. Longer words that are a power of two and a multiple of a byte and that would hold two instructions per word could also be considered from the above standpoint (24 bit words, 28 bit words or 32 bit words). This may result in problems in trying to pack data into the word for efficient byte manipulation. However, the most important point is that for the requirements considered for the given space missions, the majority of the words tend to be 12 to 16 bits and as a result a processor with a considerably longer word length can result in inefficiencies of storing data. From the above it can then be seen that there appear to be no real gains from a long word and there are some losses.

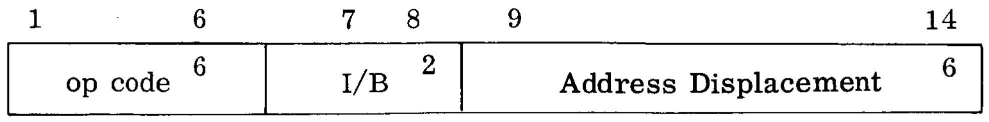
Because of the considerations given earlier a 12-bit instruction word has been eliminated as a possibility. It does not seem to provide sufficient flexibility to save storage over a longer word length. In fact, it appears that it would require considerably more storage primarily due to the addressing and short bank problems.

14-Bit Instruction Word

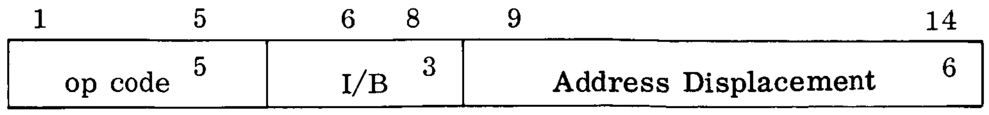
Two examples of the 14 bit instruction word are given in Figure 8-3. Figure 8-3a shows 6 bits used for the op-code. This should easily be sufficient to offer instructions to take advantage of the multiple accumulators (as many as four would be practical), to provide byte manipulation instructions, and even to provide some complex macros. Two bits are provided for index/banking using the same scheme as discussed in relation to figure 8-2. The address section of the instruction word, provides six bits for a 64 word bank. This bank should be sufficiently long for some flexibility since full length bank/index registers are used. However, since only a maximum of 4 index bank registers are available from the I/B bits, this 64 word bank could provide some inefficiencies.

The addition of more index/bank registers could alleviate much of the possible addressing problem due to the 64 word bank. Such a scheme is shown in the instruction word of Figure 8-3b. Here the op-code bits have been decreased from 6 to 5, but the I/B bits have been increased to three to offer the usage schemes shown in Figure 8-4. There are clearly other possibilities of using 3 I/B bits, but those shown in Figure 8-4 offer the most flexibility. The advantage of the scheme in Figure 8-4a over than shown in Figure 8-4b is simply the availability of more total registers that can be used for index-banking purposes. However, Figure 8-3b offers 5 registers and a considerable amount of flexibility in terms of multiple indexing. In particular the index registers do not have to be adjusted every time a bank register is changed. (This is the case in the scheme in Figure 8-4a). Because of this multiple indexing flexibility and the possibility that it may save some storage, the scheme in Figure 8-4b would be chosen. The instruction word in Figure 8-3b also has six address bits for a 64 word bank.

An explicit decision to choose between the instruction words shown in Figure 8-3a and 8-3b cannot be made until investigation into the use of various op-codes is carried out. After this a relative evaluation of the use of an additional bit for a 6-bit op-code or for a three bit I/B specification can be evaluated so that either Figure 8-3a or 8-3b can be chosen for a 14 bit instruction word. The question of course would center around which scheme provided the most storage savings; however, there are additional considerations, such as, the ease of programming, etc.



8-3a



8-3b

Figure 8-3. 14-bit Instruction Word

0 0 0	B
0 0 1	B + T_1
0 1 0	B + T_2
.	.
.	.
.	.
.	.
1 1 1	B + T_7

a.

0 0 0	B_1
0 0 1	$B_1 + T_1$
0 1 0	$B_1 + T_2$
0 1 1	$B_1 + T_3$
1 0 0	B_2
1 0 1	$B_2 + T_1$
1 1 0	$B_2 + T_2$
1 1 1	$B_2 + T_3$

b.

Figure 8-4. Use of Three I/B Bits

A 14-bit word may be an efficient choice for the computation system in the space missions under consideration, however, it does have some inefficiency problems due to its relatively short length. For example, double precision operations of 28 bits will not be sufficient for some navigation and guidance systems. As a result, triple precision will be required, but a triple precision word containing 42 bits will offer greater accuracy than that required and consequently will waste some amount of data storage area. In addition, the use of triple precision will require triple precision software to be added. There is also some question as to whether a seven bit byte will be sufficient for byte manipulations in the scientific experiments.

There is, therefore, some push toward a 16-bit instruction word due to additional flexibility in the instruction word, the use of 8 bit bytes, and more flexibility in bit manipulation. The latter point will be made clear by a renewed consideration of the discussion of byte manipulation given earlier. If a considerable amount of bit manipulation is necessary in the computations (in other words manipulation of bytes that can vary in length from one bit to eight or more bits), the use of an instruction word with a number of bits that is a power of two would be useful. These varying length bytes can then be packed into words and can be accessed by instructions by simple shifts of the address in a fashion similar to that for the half word bytes discussed earlier. All that is necessary is to place the address of the first word in a list in an index register and then to address all varying length bytes relative to this initial address by using an address in the instruction word that represents the bit number in the list. For example, a 16-bit word would simply require that the bit address be shifted four positions to the right and then indexed with the initial word address in the list. This would give a word address of the word that contained the required bits. The four bits that were shifted right would be saved and used to choose the particular starting bit in the chosen word. (The bit address can also be indexed if desired.)

Clearly, if a fairly large amount of bit manipulation was necessary, a word size that is a power of two could provide substantial storage savings in that simple instructions could easily be included to automatically carry out the bit manipulations. For example, a single instruction to load the accumulator with some desired byte would simply pick up the address bits from the instruction word, shift them four positions to the right, save them, index the shifted address with the register specifying the initial address in the word, pickup the word, and then left adjust it to the specified starting bit location given in the instruction. Additional instructions to add another set of bits to the ones that were loaded could also be implemented. These byte manipulation instructions would take a reasonable amount of time; however, they would only require a very small number of instructions for very complex manipulations.

The actual amount of byte manipulation required in the programs would have to be extensively investigated before the above discussion could be used as a strong reason for choosing a 16-bit word over a 14-bit word. However, there are also a number of additional points to be considered when trying to choose a 16 or 14 bit instruction word for the cell processor and memory. A word size increase from 14 to 16 bits would increase the number of memory bits 14 percent if the same number of total words were required. Therefore, in order for the 16-bit word to be more efficient in terms of storage savings than the 14 bit word the additional features and flexibility gained with 16 bits would have to make up 14 percent or more of the memory. The factors contributing to a decrease in the number of memory words with a 16-bit word are the following: less instructions would be required due to the ability for additional indexing or longer banks and/or additional op-codes, there would be no requirement for triple precision operations for data storage with a 16-bit word, and there may be more flexibility of byte storage as discussed above. A precise comparison of the storage usage of a 14- and 16-bit word cannot be carried out until the requirements are explicitly specified in the future. However, a rough evaluation seems to show that there would not be a sizable memory difference between the two approaches since the additional features possible with a 16-bit word would off-set the increase in bits per word by a reduction in words required. As a result, a 16-bit word will be chosen for the present design of the distributed processor cell since it will offer additional flexibility in terms of meeting a variety of requirements while providing somewhat greater programming ease. If in the future when a distributed processor is to be designed explicitly for a specified mission or set of missions and the requirements can be clearly specified, the precise trade-off can be carried out to decide between a 14- and 16-bit word. The features used in a 16-bit word are given in the next paragraph.

16-Bit Word

Two useful instruction word formats for a 16 bit word are shown in Figure 8-5. Clearly, there are other formats that are possible, but the two chosen appear to be the most applicable to the space mission requirements. One other variation that would use seven op-code bits and a lesser number of address or I/B bits may be of some interest. The additional op-code bit could be used to take full advantage of multiple accumulators (four or more) and to provide an extensive set of macros to try to save storage. However, a preliminary evaluation of the instruction set and macros seems to indicate that a six bit op-code would be sufficient; as a result, only the two 6-bit op-code formats are shown in Figure 8-5. The instruction word in Figure 8-5a uses 6 op-code bits, three bits for index banking (either of the schemes shown in Figure 8-4 could be chosen), and a 7-bit bank. The seven bit bank should provide very few inefficiencies in terms of requiring load bank commands for jumping from program to program or from one data bank to another. The instruction word in Figure 8-5b uses a 6-bit banked address and uses the additional bit to obtain 4 I/B bits. These bits can then be used for two bank registers and any one of seven index registers. This gives a powerful banking indexing and multiple indexing capability, such that a 64 word bank may cause very few inefficiencies. Further evaluation is certainly necessary to obtain an accurate tradeoff between these two 16 bit instruction word formats. Therefore, a decision between the two will not be made at this time. It should also be pointed out here that a number of instructions will be added to the op-code list in order to provide a reasonably flexible byte manipulation capability in each cell. As mentioned earlier, these will not require extensive processor hardware when a 16 bit instruction word is used (a power of 2).

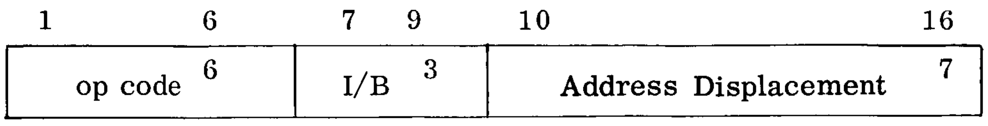
18-Bit Word

An eighteen bit instruction word with the format shown in Figure 8-6 could also be considered. However the addition of a longer bank and more op codes would not save enough memory bits to warrant the increase from 16 to 18 bits. This can be realized since the 16 bit word has very little memory restriction from either lack of op codes or bank size. The eighteen bit word would allow bigger bytes, but it provides a less flexible word size for bit manipulation (word not a power of 2). The primary advantage of this larger word would be in terms of speed increases both due to slightly less double precision operations and to a larger and more flexible instruction set. (The instruction set for the 16-bit word will not contain a number of instructions aimed at speed savings.) Therefore since storage is of primary concern in the distributed processor, it is felt that an 18 bit word is not necessary.

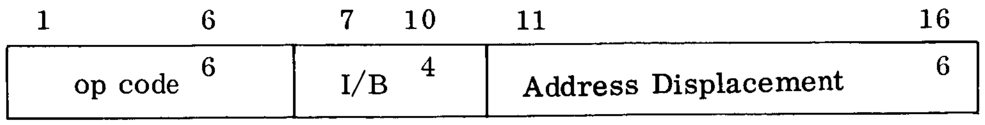
8.1.3 Control Hardware

The control hardware in the cell's processor section could be implemented with MOS gating or with a microprogrammed control unit. The primary advantages of a microprogrammed control unit are: ease of changing the instruction set by replacing the unit (e.g., replacing a diode array fixed memory wafer with another wafer), ease of design and implementation of the instruction set, and a relatively easy unit to checkout. The latter advantage may be realized since instead of complicated gating signals and combinations of signals spread throughout the processor the microprogrammed unit can be considered to be a black box with a finite fixed set of inputs giving a finite fixed set of outputs; therefore, it can be checked out by sequencing through a set of inputs that checks each memory location.

The distributed processor integrates memory and processing on a single wafer; so a primary consideration in constructing the control unit is its usage of wafer area and its affect on the yield of the processor section of the cell. A control unit constructed from MOS gating could take good advantage of redundant logic terms and could also be spread throughout the processor section of the wafer thus providing efficient gate utilization and short interconnection lines. (Note that the microprogrammed control unit would require a considerable number of long control lines.) Both of the above points would enable the gating control unit to use considerably less area than the microprogrammed unit. Therefore even though the microprogrammed control unit offers the advantages stated above a MOS gating network will be chosen for the processor section control unit in order to minimize control area. This decision can certainly be changed in the future, particularly if it is evaluated that the



5a.



Bit 7 = 0 - B_1

Bit 7 = 1 - B_2

Bit 8 - 10 = 0 0 0

0 0 1 - T_1

0 1 0 - T_2

.
.
.

1 1 1 - T_7

5b.

Figure 8-5. 16-bit Instruction Words

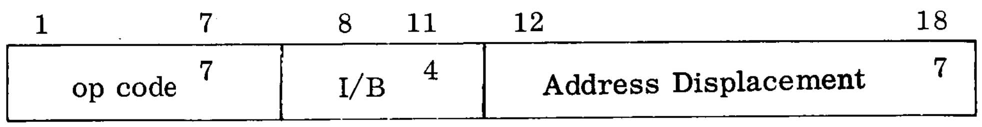


Figure 8-6. 18-bit Instruction Word

instruction set may be changed often in the distributed processor in order to apply the system to a variety of missions. (In order to change the instruction set with MOS gating the processor area of the wafer mask must be remade; however if a diode array fixed memory control unit is included in the processor a new one-zero pattern must simply be encoded into the mask.)

8.2 PROCESSOR HARDWARE

The design of the processor section is not complete at this time. However, a general description will be given here along with some specific characteristics that have been decided on thus far in order to further describe what a cell consists of. Figure 8-7 shows a preliminary design of the processor section of the cell and the memory registers used as part of the processor. The various parts of this processor will be discussed below.

8.2.1 Memory, Operational, and Communication Registers

The use of the operational registers is very similar to that described in reference 2. The description from this reference, updated for the distributed processor, is given below:

U — Hardware Upper Accumulator: The hardware upper accumulator is used to hold one of the memory upper accumulators in any operation that uses them.

L — Lower Accumulator: The lower accumulator is used primarily in multiply, divide, and double precision operations to hold the lower half of a data word. This accumulator, and U have a one-bit extension onto their sixteen bits in order to hold the overflow carries which may be generated in the multiply operation.

U_1, U_2, U_3, U_4 — Memory Upper Accumulators: The memory upper accumulators are the primary arithmetic and logical registers in single precision operations. They are also used to hold the upper half of data in double precision operations and to hold and manipulate data in shift and register operations.

P — Program Counter: This register is used to sequence the flow of control in the processor. It is not only used to access instructions but also to provide memory addresses for interrupt status word storage. It must therefore be connected both to the ALTU and to the memory interface lines.

MAR — Memory Address Register: This register holds the memory address for operand memory cycles. It is loaded with the address displacement from the instruction word, B_1 or B_2 is added to it and, if indicated, one of the index registers is also added to it. This register is necessary since the B and T registers are in memory and must therefore enter the processor through MB; as a result MB can not be used to hold the operand addresses.

MB — Memory Buffer: The memory buffer receives data and instructions from the memory, sends data to the memory, holds the divisor in divide operations, and the multiplicand in multiply operations. It also holds one of the operands in all other arithmetic and logical operations with the memory. In addition to the above tasks since the MB receives all instructions it keeps many of these bits for the instruction decoding and operation. For example, it holds the B bit for address generation, the register to be shifted in a shift operation and one of the registers to be operated on in register operations.

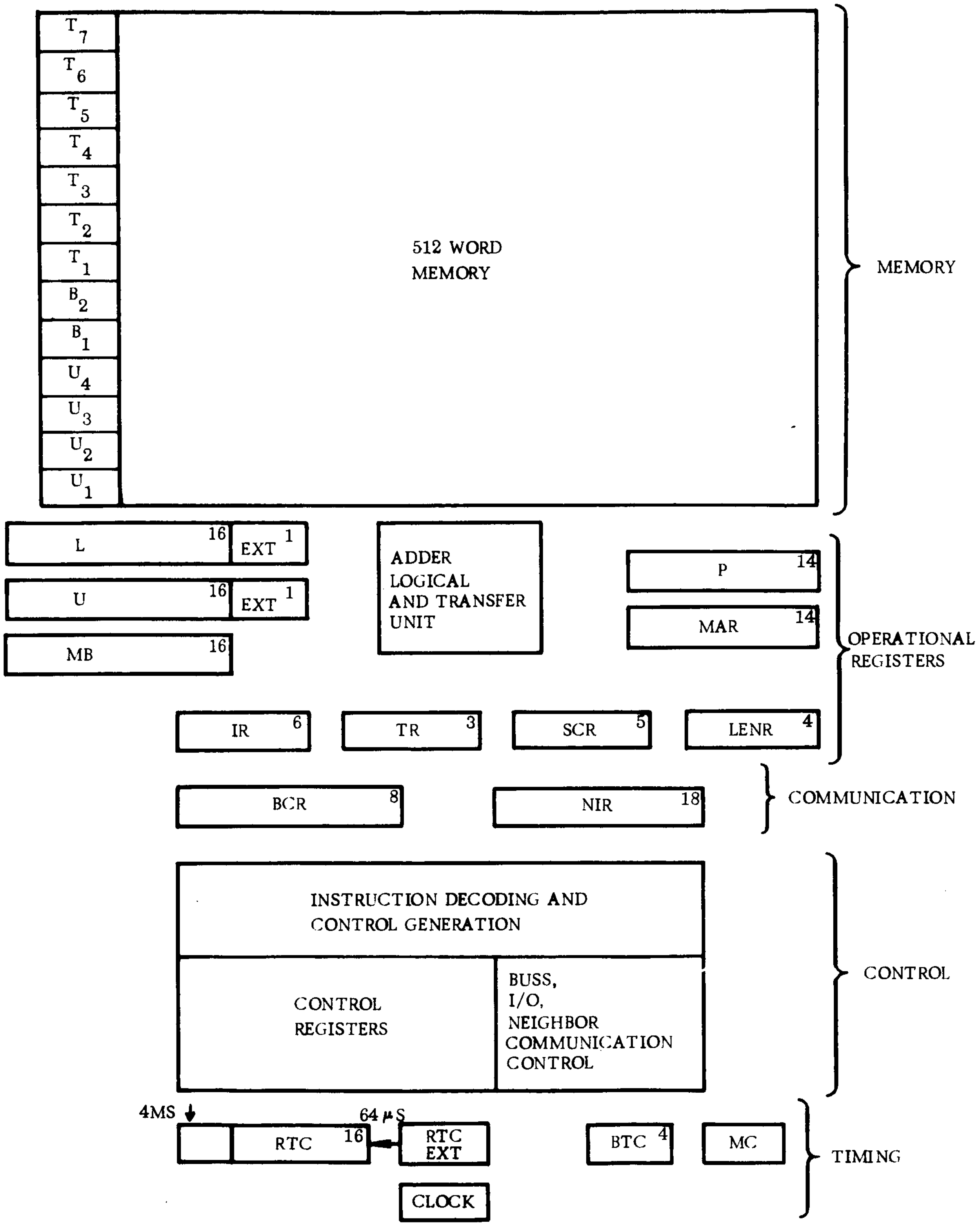


Figure 8-7. Processor Section

B_1, B_2 - "B" Memory Index/Bank Registers: These registers hold both index and bank values for address calculation and looping control. One of these two registers, indicated by the B bit, is added to the address displacement for all operand address calculations

T_1 to T_n - "T_n" Memory Index/Bank Registers: These registers have the same functions as the B registers. The only difference is that operand addresses can be generated without adding any T_n register to B plus the address displacement. (Tag 000 specified no indexing with the T_n registers.) Seven registers are shown in the figure. This would be the case if a 6 bit address decrement is used. If a 7 bit address decrement is used in the instruction word then only three T_n registers will exist. As mentioned earlier, this choice will be made later.

ALTU — Adder, Logical, and Transfer Unit: This unit contains all the circuitry for carrying out arithmetic and logical operations including comparisons. It also provides for transfers amongst all the hardware registers and detection of overflows.

IR — Instruction Register: The instruction register holds the six bit op code throughout the instruction execution.

TR — Tag Register: The tag register holds the T_n bits of the instructions. It is necessary so that B plus the address displacement can be generated, stored in MAR, and then added to T_n prior to an operand cycle. This register also holds one of the register addresses in register operations. It also holds a two bit op code extension for byte operations. This register will be 2 or 3 bits long depending on which 16 bit instruction format is selected.

SCR — Shift Count Register: This register holds the shift count for shift commands, and for setting up bits in byte manipulation operations. It is counted down to zero by one count for each shift. The register can be loaded from the ALTU in addition to the MB since shift counts may be indexed prior to being loaded into SCR for execution.

LENR — Length Register: This register is used for byte instructions to specify the length of the byte that is being used. It is also used to hold the op code extension bits in shift and register instructions.

BCR — Buss Communication Register: This register receives the present word from the inter-cell buss. (Present indications are that an 8 bit bus will be used). The communication control will then decide if the word is of interest to this cell. The register is also used to place a word on the buss.

NIR — Neighbor and I/O Communication Register: This register is the buffer for the serial neighbor to neighbor communication lines and also for the serial I/O line from a cell.

8.2.2 Accumulator Mechanization

A somewhat more detailed discussion of the accumulators is necessary in order to understand their use. There are a number of methods of handling the four accumulators in each cell. One method would assign tag bit combinations to each accumulator such that U_1 is the hardware accumulator and U_2, U_3 and U_4 are the memory accumulators. Each instruction would then specify one of these accumulators and cause its contents to be exchanged with those of the hardware accumulator, U_1, for execution of the operation. At the conclusion of the operation the hardware accumulator would keep its current value so that any further operations of this value would be carried out by specifying U_1. This scheme uses the minimum amount of processor hardware for handling the accumulators, but it makes it difficult for the programmer to keep track of information that was initially or subsequently stored in one of the accumulators. It would also be necessary at the ends of a branch to restore the accumulators to some specified ordering of information that is consistent between the branches. Because of the above two disadvantages, this scheme was not chosen.

A second scheme would replace the named accumulator into its original location at the completion of each operation. This means that unless the hardware accumulator, U_1, was in operation, a final exchange of the present value of U_1 and the orignal value of U_1 in one of the memory accumulators would have to be made. The main disadvantage of this approach is that it requires two extra memory cycles for any accumulator operation that does not use U_1. This extra time could be considerable since when an accumulator is brought into execution it is generally used for a few instructions; as a result each operation would require two extra memory cycles. For this reason this approach was not chosen.

A third approach as follows is also possible. Four sets of tag bits are held in the processor to specify the tag associated with the hardware or memory accumulator positions. (In this scheme U_1, U_2, U_3, and U_4 can be in any of three memory positions or the hardware position.) Each instruction operating with an accumulator simply specifies the tag bits of the accumulator that it would like to use. This accumulator is then found by an automatic comparison to the accumulator tag bits, HA1, MA2, MA3, and MA4. The specified accumulator is then loaded into the processor accumulator position (if it is not already there) and the accumulator tag bits are updated to reflect the present locations of U_1, U_2, U_3, and U_4. (When the machine is started, these tag bits must be loaded into HA1, MA2, MA3, and MA4 in any order.) This scheme requires slightly more processor hardware than the other schemes mentioned, but it has the advantage of leaving the last accumulator referenced in the hardware accumulator position. Therefore only the first reference to a new accumulator requires an extra memory cycle (the exchange of accumulators can be carried out in one memory cycle). It should be noted that the accumulator tag bits in the processor must be stored after an interrupt in order to enable proper restarting of an interrupted program.

A fourth approach similar to the last scheme is described below. Four locations in memory, as shown in Figure 8-7, are used to hold U_1 to U_4. Whatever accumulator is referenced by the instruction word tag bits, is placed in the hardware accumulator position and the present contents of the hardware accumulator are returned to their proper memory position. Two control bits HA1 are necessary so that the accumulator presently in hardware can be specified and compared to the tag bits in the instruction.

Clearly, if the present hardware accumulator is specified no memory access is required. This scheme then accomplishes the same operation as the last scheme (it leaves the last referenced accumulator in the hardware position) but uses slightly less processor hardware for control and one more memory location. It also requires only one additional memory cycle when an accumulator from memory is specified since the present accumulator value should be able to be replaced in its proper memory position and the new accumulator picked up all in one memory cycle. At the same time the HA bits will be updated to the new accumulator tag. This scheme was selected over the third approach described above because it should actually require less total hardware usage. It requires less control and register hardware in the processing section due to requiring only the HA tag bits. More important though is that less bits will have to be stored upon an interrupt which may actually make up for the extra memory location used for the accumulator.

8.2.3 Timing

A real time clock (RTC) and real time clock extension will be included in each cell in order to provide interrupts to enable scheduling of real time programs. This clock is basically the same as that discussed in reference 2, section 6.1.1; as a result it will not be discussed here. The only new point is the fact that the system clock here is not yet specified so that the length of the RTC Ext. cannot be set; however the clock time (or bit time) will probably be on the order of 2μs so that the scheme in Figure 8-7 with a 5-bit RTC. Ext. would be sufficient. The clock can be set and read by two instructions. The bit time counter (BTC) will be four bits and will be incremented by the clock. It will in turn increment the mode counter (MC) that is used to keep track of the various phases of long instructions. Since all the instructions have not yet been specified, the length of this counter will not be given. (It must be long enough to handle the longest instruction.) This timing hardware is also discussed in reference 2, section 6.1.1.

8.3 INSTRUCTION SET

The specification of the instruction set is nearly complete and will be included in the next report. A sample instruction execution sequence will be given here to illustrate the use of the hardware previously described:

The execution of the add instruction will be given below; the following definitions will be used:

(M):	Contents of Addressed Memory Position
U:	Any of the upper accumulators
m:	Address displacement
MAR:	Memory Address Register
(P):	Contents of program counter
MB:	Memory Buffer Register

B:	One of the B index/bank registers
T_n:	One of the T_n index/bank registers
U_h:	Hardware upper accumulator
U_m:	Any of the memory upper accumulators
$\rightarrow$:	Replaces
INSTRUCTION:	ADD
ADU:	$(M) + U \rightarrow U$

This is executed as follows with no indexing and with U located in the hardware location:

Instruction access: $m \rightarrow MAR$, $(P) \rightarrow MB$	1 memory cycle
Bank access: $B + MAR \rightarrow MAR$	1 memory cycle
Operand access and execution: $(M) + U \rightarrow U$	1 memory cycle
	3 memory cycles

It is executed as follows with indexing and with U located in one of the memory accumulator positions:

Instruction access: $m \rightarrow MAR$, $(P) \rightarrow MB$	1 memory cycle
Bank access: $B + MAR \rightarrow MAR$	1 memory cycle
Index Access: $T_n + MAR \rightarrow MAR$	1 memory cycle
Accumulator access: $U_h \rightarrow U_m$ (old accumulator put in its location)	1 memory cycle
$U_m \rightarrow U_h$ (new accumulator picked up)	
Operand access and execution: $(M) + U \rightarrow U$	1 memory cycle
	5 memory cycles

GLOSSARY

Calculated Address — An address calculated by a cell using a bank (base) register, index register if one is specified, and the displacement field from the instruction.

Cell Bus — The communication wires or lines connecting all the cells in a group.

CB — Control Byte

CC — Controller Cell

Control Byte — The first 8 bits of a control word. The control line, one of the lines that make up the inter cell bus, is always set while this byte is being transmitted.

Control Word — One or more bytes that are sent by the controller cell to control other cells. The first byte of this word is always the control byte.

D16 — A 16 bit data word that follows an instruction. The term also refers to a GC modifier that specifies a D16 instruction modification.

D32 — The same as a D16, only the data word is 32 bits in length instead of 16.

DS — A form of instruction modification that specifies a list of 16 bit data words. A given address is always present, and precedes the data and follows the instruction.

Dependent State — A cell that responds to GC level instructions and to cell addresses.

Effective Address — The address used by the cell to specify which word of a cell's memory is to be used.

GC — Global control instructions. These are level and format control instructions.

Given Address — An address that is specified by an instruction modifier. The address always follows the instruction that has been modified.

I — Immediate

Identification Register — The register in a cell containing the cell address. Cells are given unique cell address by the controller cell.

Immediate — One form of instruction modification where the data to be used is the displacement field of the instruction.

Independent Cell — A cell whose state prevents the processor from responding to GC level instructions sent over the inter cell bus. (Independent cells use local communications.)

Instruction — An operation, such as add, multiply, in a program. The categories of instructions are given in Table 5-2.

Instruction Modification — An instruction is preceded by a special instruction, called a GC modifier, that modifies the normal operations performed by the instruction.

Level Register — The register in a cell containing the level number for this cell.

Respond (to a Control Word) — The cell receives all bytes of a control word. A cell responds to a control word when the identification register and control byte address are equal, or, for dependent cells, the level number in the cell and control byte level are equal. In all other cases, the cell will receive only the first byte of a control word.

State — A cell exists functionally in one of seven states. A state defines how the processor shall interpret instructions and where the instructions shall be fetched. Table 5-1 lists the seven states.

REFERENCES

1. Study of Spaceborne Multiprocessing, first quarterly report, phase II, C6-1476.13/33; Autonetics, Anaheim, California.

2. Study of Spaceborne Multiprocessing, final report, phase I, C6-1476.10/33; Autonetics, Anaheim, California.

3. Study of Spaceborne Multiprocessing, third quarterly report, C6-1476.8/33; Autonetics, Anaheim, California.

4. Volden, Jack E., "The Cordic Computing Technique". Proceedings of the Western Joint Computer Conference, March 1959.

5. Hartig, David, "Microelectronic Digital Stabilization Computer", Bureau of Naval Weapons Symposium for Rotating and Static Components, April 22, 1964.

6. Results of Multi-Accumulator Study for Next Generation Computer; Internal Report; Autonetics, Anaheim, California.

7. Computer Memory Banking Study; Internal Report; Autonetics, Anaheim, California