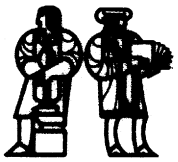
LABORATORY FOR
COMPUTER SCIENCE
*(formerly Project MAC)*



MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TR-186

# A STRUCTURE MEMORY FOR DATA FLOW COMPUTERS

William B. Ackerman

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under Contract No. N00014-75-C-0661

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

*This blank page was inserted to preserve pagination.*

MIT/LCS/TR-186

# A STRUCTURE MEMORY FOR DATA FLOW COMPUTERS

by

WILLIAM B. ACKERMAN

August 1977

Massachusetts Institute of Technology

Laboratory for Computer Science
(formerly Project MAC)

Cambridge Massachusetts 02139

# A STRUCTURE MEMORY FOR DATA FLOW COMPUTERS

by

William B. Ackerman

Submitted to the Department of Electrical Engineering and Computer Science on August 26, 1977 in partial fulfillment of the requirements for the degree of Master of Science.

## ABSTRACT

A data flow computer is one which achieves enormous concurrency of instruction execution through a machine architecture that acts directly on a data dependency graph of the program. To handle arrays and data structures effectively, a data flow computer must have access to a memory system which can handle large numbers of concurrent transactions. This thesis presents a design for such a memory. A "cache" mechanism is presented for improving the performance of the system, and a mechanism is given for using sequential-access devices such as shift registers as the memory medium. The memory system design uses the "packet communication" concept, in which the components of the system communicate only through the transmission of fixed size "packets" of data.

THESIS SUPERVISOR: Jack B. Dennis

TITLE: Professor of Computer Science and Engineering

## ACKNOWLEDGMENTS

I wish to thank Professor Jack Dennis for his encouragement and support through this research and for providing an intellectually stimulating environment in the Computation Structures Group. I would like to thank Glen Mraniker and Lynn Mentz for their helpful comments on parts of this thesis. The Laboratory for Computer Science provided facilities for the preparation of this thesis.

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract no. N00014-75-C-0661.

# TABLE OF CONTENTS


| | |
|---|---|
| Title page | 1 |
| Abstract | 2 |
| Acknowledgments | 3 |
| Table of Contents | 4 |
| 0.0 Introduction | 5 |
| 1.0 Data Flow Computers | 6 |
| 1.1 Data Structures | 12 |
| 1.2 The Structure Controller | 17 |
| 2.0 Specifications of Packet Systems | 24 |
| 2.1 Nondeterminacy | 33 |
| 2.2 Packet Acknowledgments and Safety | 45 |
| 2.3 Latency, Arbitrators, Distributors, and Allocators | 51 |
| 3.0 The Basic Memory Module | 59 |
| 3.1 Horizontal Interconnections of "MM" Systems | 66 |
| 3.2 Vertical Composition and the Cache Module | 70 |
| 4.0 Implementation of MM Using a "Rotating" Device | 83 |
| 5.0 Structure Controller Design Considerations | 95 |
| 6.0 The Deadlock Problem | 104 |
| 7.0 Suggestions for Further Research | 108 |
| References | 109 |
| Appendix I | 111 |
| Appendix II | 114 |
| Appendix III | 121 |

## 0.0 INTRODUCTION

A data flow computer is a machine with architecture radically different from that of existing computers. It can perform computations simultaneously on many different parts of a program. A typical data flow computer has many arithmetic processors, and can utilize all of them simultaneously, each executing a different instruction.

To handle arrays and other data structures, a data flow computer must have a data structure processing facility and memory that has a similar facility to perform many operations concurrently. Such a data structure memory is the subject of this thesis.

A data flow computer owes its great speed to its ability to perform many operations at once, even though each individual operation is no faster than on a conventional computer. The same is true of the memory. The memory to be presented here has a retrieval delay just as great as conventional memories, since no new circuit technology will be proposed. However, it has an enormous data transfer rate because of its ability to handle concurrent transactions. This concurrency is made possible by an unusual type of interface called packet communication.

Section 1 of this thesis is an overview of data flow computers and the type of memory that such a computer requires for structure processing. Section 2 is a treatment of packet communication systems, showing how their behavior is defined. In section 3 the basic memory unit is described, along with a "cache" mechanism and an "interleaving" method to improve its performance. In section 4 an implementation of the memory using shift registers or magnetic disks will be given, showing how the disadvantages of such devices can be overcome through the use of packet communication. Section 5 examines some aspects of the processing unit that uses the memory, and section 6 examines the "deadlock" problem and the cost of overcoming it. Section 7 presents suggestions for future research.

## 1.0 DATA FLOW COMPUTERS

As the need increases for ever faster computers, one technique for improving performance that has drawn considerable interest in the last few years is a radically new design known as a data flow computer [6] [7] [11] [18]. A conventional computer has only one locus of control, that is, one point in the program at any given instant at which instructions are executed. Ability to execute more than one instruction at a time can improve performance significantly, and some computers use an instruction lookahead to achieve this [3] [9]. However, the benefits of lookahead methods are limited, and such computers are enormously complex. Other attempts to increase instruction concurrency include "array processors" [16], but such machines are inflexible and extremely difficult to program.

A data flow computer achieves executional concurrency by using a different internal representation of the source program. Instead of representing the program as a list of instructions to be executed in a particular order, the program is represented as a data flow schema. A data flow schema is a directed graph whose nodes represent instructions and whose arcs show the data dependence among instructions. The order of instruction execution is determined solely by the data dependence - an instruction is executed when all of its data sources have produced results and all of its destinations are ready to receive data. This allows many instructions throughout the program to be executed simultaneously.

The data in a data flow program can be modeled by "tokens" that reside on the arcs of the graph. Each arc may contain at most one token. The execution rule for most instructions is as follows:

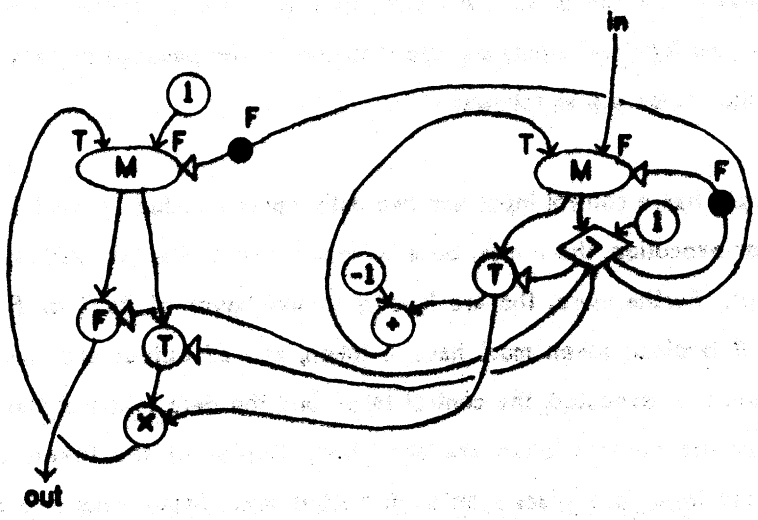
> An instruction (other than a merge or gate) is ready for execution whenever all of its input arcs contain tokens and all of its output arcs are empty. When an instruction is executed, the tokens on the input arcs are absorbed. The function denoted by the instruction is computed, using the values in the absorbed tokens as input data. A token containing the function value is placed on each output arc.

There are a number of ways of handling decisions and iteration control. Perhaps the simplest is the use of special instructions M, T, and F. These receive a boolean value on one input (the "control" input) and use it to control the passage of data from another input. Their execution rules are as follows:

> The M (merge) has a control input and two data inputs labelled "T" and "F". To be ready for execution, there must be a boolean token on the arc leading to its control input. Furthermore, the arc leading to whichever of its T or F input matches that boolean token must have a token, and all output arcs must be empty. When it is executed, the control token and the data token at the input indicated by the control token are absorbed. Copies of the token at the selected data input are placed on each output arc. Input tokens are not required at the non-selected data input, and if any are present they are not absorbed.
>
> The T (true gate) and F (false gate) instructions have a control input and a data input. They are ready for execution whenever both input arcs contain tokens and all output arcs are empty. When they are executed, the inputs are absorbed. If the control input matches the name of the instruction, copies of the data input are placed on the output arcs. If not, no tokens are placed on the output arcs.

Constants can be generated through the use of functions of no arguments. An instruction to perform such a function has no input arcs, so, in accordance with the execution rule, it places tokens on its output arc as fast as they are removed.

Here is an example of a data flow scheme to compute the factorial function:




Boolean inputs to M, T, and F instructions are drawn as open arrows. Tokens existing in the initial configuration of the program are drawn as filled-in circles.

The behavior of a data flow scheme under the execution rules has a very important property - it is determinate. This means that the output of the program is determined only by the input, and is independent of the timing of instruction executions. All runs of such a program with the same data will yield the same results. Determinacy follows from the facts that

(1) Each instruction produces a result which is a function only of the values of its input tokens, that is, each node of the scheme is determinate.

(2) The value of a token does not change in any way while it resides on an arc.

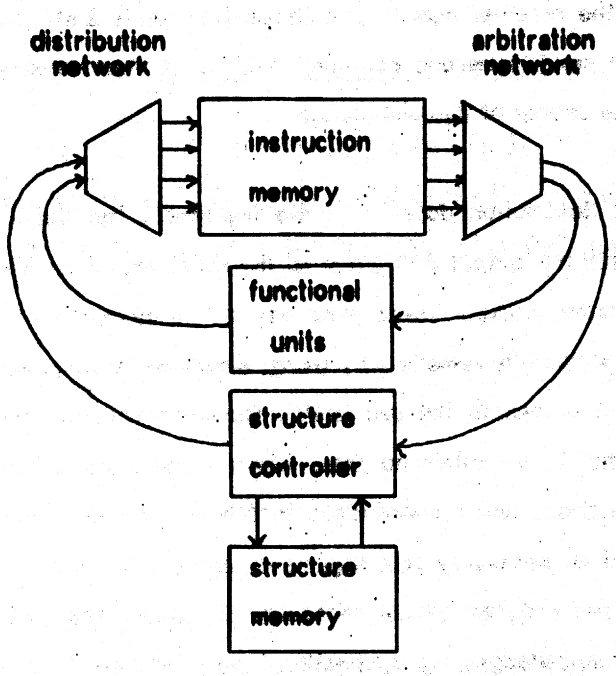
(3) The execution rules, and fact (2) above, qualify the scheme as a valid interconnection of autonomous communicating systems.

It is an established result that such an interconnection of determinate systems is determinate [1] [14].

### 1.0.1 DATA FLOW COMPUTER ARCHITECTURE

The memory system and structure processor that are the subject of this thesis are intended to be part of a computer of the type described by Dennis and Misunas [6] [7]. Such a computer is composed of units which use packet communication [8] for transfer of data. The only means of data transmission among these units is the transmission of fixed size messages called packets. There is no clock or synchronizing information.

The four main parts of the data flow computer are the instruction memory, arbitration network, functional units, and distribution network. For structure processing, the structure controller and structure memory are added.




To execute a data flow program, its schema is encoded into the instruction memory. Each cell of the memory contains one instruction of the schema. At the time the program is loaded, each cell is filled with the operation code (arithmetic operation, merge,

structure operation, etc.) and the address of its destinations. The latter are the cells to which outgoing arcs point. The instruction cells also have receiver registers to contain incoming "tokens". When all necessary receiver registers become full, an instruction cell emits an operation packet, consisting of its operation code, the data from the receiver registers, and the destination addresses.

Any given program has a great number of instruction cells, each sending operation packets only occasionally. These streams of packets are merged by the arbitration network into a small number of dense streams. The packets coming out of the arbitration network are sorted according to operation code and sent to the appropriate functional units. In the case of structure processing instructions, they are sent to the structure controller. The functional units or structure controller perform the indicated operation and form, for each destination, a result packet consisting of the destination address and a copy of the actual result. The result packets go to the distribution network, where they are sorted by address and sent to the appropriate receiver register of the appropriate instruction cell. (The destination address includes the receiver number.) If the instruction is a structure operation, the structure controller may send numerous command packets to the memory and receive result packets back during the course of its computation.

The preceding description does not quite implement the execution rule: An instruction cell should wait until its "output arcs", that is, the receivers of its destinations, are empty before issuing an operation packet. There is no way for an instruction cell to "see" its destinations' receivers. The problem is remedied by using, where necessary, acknowledgment tokens sent from a cell's destinations to the cell itself. The acknowledges are treated like invisible arguments, except that they contain no data. When a cell is executed, it may send result packets to some destinations and acknowledges to others. A cell is not ready to be executed until it has received all necessary real arguments and all necessary acknowledges. Acknowledges are placed in the program where necessary to ensure that, when a cell has received all arguments and acknowledges, its destinations' receiver registers will be empty. These acknowledges should not be confused with the packet acknowledges to be developed later.

A constant need not be implemented as a separate node of the data flow schema. It can simply be loaded into the receiver register of the instruction cell that uses it, and marked in such a way that the instruction cell knows that that register is always full.

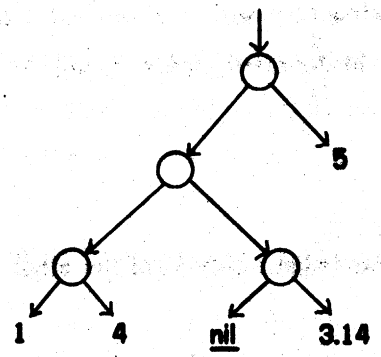
An additional part of the data flow computer, not shown in the preceding diagram, is the host computer. This is a computer of conventional design, which has access to the memory units and control functions of the data flow computer. It is used for diagnostic testing and for initial loading of the instruction memory and structure memory. It does not participate in the actual data flow computation.

## 1.1 DATA STRUCTURES

In order to handle arrays and data structures in a data flow computer, it is in most cases necessary to allow single tokens to have entire structures as their values. (Some programs which use arrays of fixed size, such as Fourier transforms and other signal processing algorithms, can make do with arrays of instructions with one token on each arc. However, this approach is impractical for very large arrays or for dynamic structures.) For this reason, we propose a data structure facility that allows tokens to have structure values. The simplest type of structure that permits full generality is the binary tree, which is recursively defined: a binary tree is an elementary "object" from some set, or is a concatenation of two binary trees. Such trees form the basis for the programming language LISP. [4] [13] For definiteness, the structures used in a data flow computer will be assumed to be binary trees.

The "elementary objects" are all data values other than structures that the computer can handle, plus the special object nil. Elementary objects thus might include integers, boolean values, reals, etc.

The principal operation on a data structure is selection. A simple selection takes a structure and a single bit. If the structure is elementary and not nil, the result of the selection is undefined. If the structure is nil, the result is nil. Otherwise, the structure is the concatenation of two structures, and the result of the selection is the first or second of these if the bit is zero or one respectively. A compound selection takes a structure and a string of bits, and gives the result of applying simple selections repeatedly, using the bits in sequence. The bit string is called the selector. Let S be the following structure:




SELECT[S, '1'] = 5 (a simple selection)

SELECT[S, '001'] = SELECT[SELECT[SELECT[S, '0'], '0'], '1'] = 4 (a compound selection)

The true "meaning" or "value" of a structure can be defined to be the set of ordered pairs of selectors that yield elementary values other than nil, along with those values. Thus the structure S denotes the set

{ <'000', 1>, <'001', 4>, <'011', 3.14>, <'1', 5> }

Nil simply denotes a substructure with no elementary items at all.

Using this definition of the meaning of a structure, there is a structure corresponding to any finite set of ordered pairs of selectors and elementary values (excluding nil) such that no selector in the set is an initial substring of another. The structure nil denotes the empty set.

SELECT[struc, sel] =

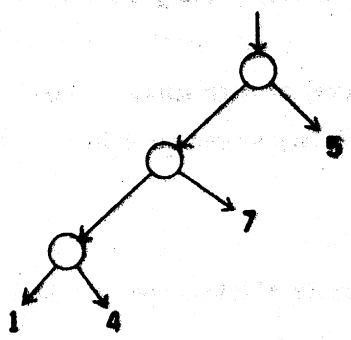
The elementary value v if struc contains the pair <sel, v>
Undefined if <s, v> ∈ struc where s is a proper initial substring of sel
The structure { <s, v> | <sel•s, v> ∈ struc } otherwise

Structures can be built with the append operation. APPEND places a given object (structure or elementary value) onto a given structure with a given selector, removing whatever substructure previously existed there. In the set-theoretic model,

APPEND[struc, new-val, sel] =

$$\begin{cases} (struc - \{ \langle s, v \rangle \mid \text{one of sel or s is an initial substring of the other} \}) \cup \{ \langle sel, new\text{-}val \rangle \} \\ \qquad \text{if new-val is elementary.} \\ (struc - \{ \langle s, v \rangle \mid \text{one of sel or s is an initial substring of the other} \}) \cup \\ \qquad \{ \langle sel \cdot s, v \rangle \mid \langle s, v \rangle \in new\text{-}val \} \quad \text{if new-val is a structure, including } \underline{nil}. \end{cases}$$

Letting S be the structure defined previously, APPEND[S, 7, '01'] is




The substructure containing nil and 3.14 disappears.

### 1.1.1 REPRESENTATION IN MEMORY

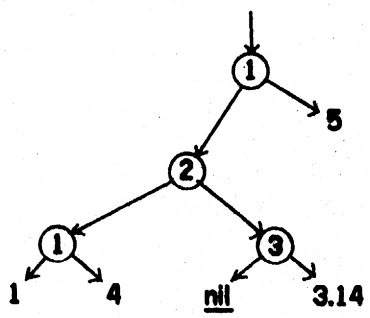
Structure can be implemented on a data flow computer in the same way that they are commonly implemented on ordinary computers - as linked lists of "cells" in a memory. An elementary object is represented by the object itself. A concatenation is represented by the address in memory of a cell containing the representations of the two substructures. In either case, a structure is represented by a small amount of information. The huge amount of information that constitutes the structure itself lies inside the memory, and the representation is merely a pointer to this. The operation of selection is quite simple. Cells are read from

memory and the appropriate halves of the data used, under control of the selection bits.

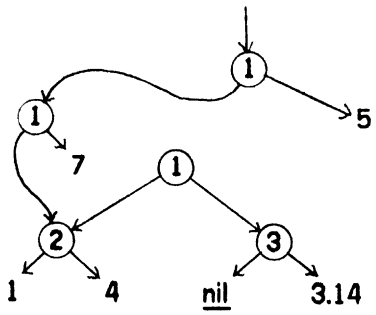
### 1.1.2 SHARING

Such an implementation leads to the possibility of a single structure in memory being shared (or partly shared) by several parts of the computation. In a data flow computer, two tokens might have the same pointer as their value. This is of course very desirable for economical memory use, but it makes the APPEND operation difficult. The problem is that modification of pointers inside the memory can change the value of structures other than the intended one, if structures have parts in common. In many programming languages, this is considered a reasonable and even desirable effect. For example, the LISP language has instructions to modify existing structures. In a data flow computer, however, this cannot be permitted for reasons of determinacy. In order for a data flow computer to be determinate, the meaning (in the set-theoretic sense given previously) of a token bearing a structure value must not change while that token resides on an arc. Since other instructions, including APPEND's, can be executed while a token resides on an arc, APPEND must never change any substructures that are shared with other structures.

In the proposed structure processing facility, each cell has a reference count which makes it easy to tell what substructures are shared. Whenever the APPEND processor is tempted to modify a cell that is shared with another structure, it makes a copy of the cell and modifies the copy instead. For example, if S is a pointer to the following structure in memory:




where the number in each node is the reference count, APPEND[S, 7, '01'] yields




The node that originally had a reference count of two may not be modified, so a copy is made, and its reference count is therefore reduced to one. The structure controller to be described in the next section will perform these tasks.

## 1.2 THE STRUCTURE CONTROLLER

In this section we will outline the behavior of a processing mechanism that uses the structure memory to provide a structure facility for the data flow computer. The basic behavior of the structure controller is that it receives operation packets from the arbitration network and delivers result packets to the distribution network. It holds the state information for structure operations in progress, and performs memory operations by sending packets to the memory and receiving packets in return.

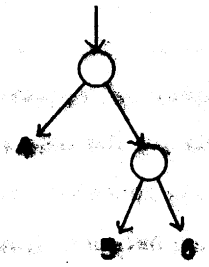
The purpose of this section is to show how the structure controller will use the memory, rather than to give a detailed specification for the structure controller. Therefore, a number of design decisions will be made arbitrarily. For the most part, the requirements of the structure memory are independent of these decisions. For example, the memory design would not change if ternary trees were used instead of binary ones.

Some aspects of the design of the structure controller will be considered in more detail in section 5.

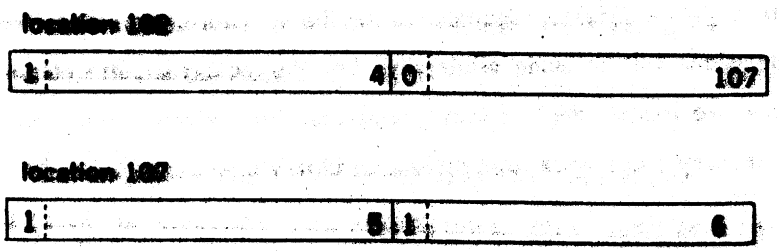
### 1.2.1 DATA FORMAT

The memory space is divided into "words" or "cells", each of which holds one node of a structure. Since the memory is used for the storage of binary trees, the words representing nonterminal nodes contain two pointers to other nodes. The convention will be made that all words of the memory will be divided into halves, called the left half and the right half. Each half has an "elem" bit bit indicates whether it contains an elementary item (terminal node) or a pointer to another word in the memory. If the bit is 1, the half word contains an elementary value. The interpretation of that half word is then the exclusive responsibility of the rest of the computer, unless it is nil. The structure controller treats any elementary value other than nil simply as a collection of bits. Any type information (integer, floating point number, character, etc.) must be encoded into the half word along with the data.

The structure graphically represented as follows:




might be realized by address 102 in the following memory configuration:




The bit at the left end of each half word is the "atom" bit.

(A different convention could be used, in which each elementary value takes an entire word instead of half a word. The two conventions are equally powerful, and differ only slightly in execution. The "half word" convention will be used for definiteness.)

### 1.2.2 MEMORY MANAGEMENT AND GARBAGE COLLECTION

All words of memory that are not part of a structure are kept in a collection of free storage lists. (There are several such lists, rather than one, in order to maintain a high rate of processing. This point will be discussed in section 5.0.5.) Whenever the structure controller needs a word in order to create a node, it takes it from one of the lists. Whenever a node is destroyed, that is, all pointers to it disappear, the word containing it is returned to a free storage list.

Each node of a structure has a reference count, which is the number of pointers to that node that exist, whether in other nodes or in the rest of the computer. (The latter includes operands waiting in instruction cells and packets in transit through the arbitration and distribution networks.) The structure controller increases or decreases the reference count of each node as pointers to it are created and destroyed. When the reference count is decreased to zero, the node disappears, so it is returned to a free storage list. Whenever this happens, any pointers that the node contained disappear, and so the reference counts of the nodes pointed to must be decreased.

The choice of a reference count strategy for memory management instead of the "mark and scan" method commonly used in LISP systems was made for three reasons:

(1) The mark and scan method requires a garbage collection operation which must find every reference to every structure. Since references exist in packets in transit, it would be necessary to stop the entire computation and wait until all packets stop moving before a garbage collection commences.

(2) The reference count is needed anyway in order to implement the copying rule efficiently. Whenever the structure controller needs to modify a node as part of an APPEND operation, it may do so safely if the reference count is one. If not, the node must be copied.

(3) The objections to the reference count method in many list processing systems, that it is difficult to recover circular lists, does not apply here. Because of the copy rule, circular lists are never created.

### 1.2.3 THE STRUCTURE OPERATIONS

The structure controller to be proposed implements the following program level operations:

SELECT(structure, selector) - The selector is a bit string of definite length. The structure is traced under control of the bits in the selector, starting with the leftmost bit. A zero bit selects the left offspring and a one bit selects the right. The item at the selected point in the structure is returned, whether it is elementary or a substructure.

APPEND(structure, object, selector) - Returns a structure similar to the given one, but having the object at the place specified by the selector. Whatever was at that place in the original structure is absent in the result. The object may be elementary or a structure. Any part of the original structure that is shared with other parts of the computation is not modified. The controller copies part or all of the original structure as necessary to be sure that this is the case.

The structure controller recognizes the special constant nil which, while elementary, is also the structure with no selectors. Nil is used as a terminal node of a structure to indicate that there are no objects beyond that point. Any part of a structure may be deleted simply by using the APPEND operation to replace it with nil, and a structure may be created by appending something to nil. It is assumed that the constant nil is explicitly available to the programmer for these purposes. The controller optimizes all structures, replacing with nil any substructure all of whose terminal nodes are nil.

There are two more operations performed implicitly by the controller. If any operation returning a structure value specifies more than one destination, the reference count of the result must be appropriately increased. Also, if any operation discards a structure value, the reference count must be decreased. It follows that the conditional operations such as true and false actors must be executed by a structure controller if the objects being switched are structures.

memory, and then use them
complicated structure operat
amount of state information
that make up the structure
datum to be appended, the
and a few pointers. The tot

There are two
of it in the tag field of comm
store any information about

fields for elementary values.
are irrelevant to the memory.
field. In practice, it might be

From the mem
associated with each word. I
the memory system, althoug
count. In a typical realization

Incoming and o
and reference count fields tha

[redacted]

description, for every inpu
eventually produce. "Even
histories, it means that, for
because a system which is e
entire output in finite time.

A description
called a functional specifica
The major problems in the
built in a certain way ob
interconnection of systems

## 2.0.2 DESCRIPTIVE SPECIFIC

Since a major

in a certain way obeys cer

reasonably formal way how

too rigid and implementation

assembled from components,

to describe the interconnect

each other. For systems th

given in terms of a progra

RTR(addr, dat

(RTR stands f

Every READ

bearing the address and the

data in the memory and ret

memory is zero.

For a given

determined. The contents

having that address, or zer

[illegible]

This specifica[illegible]
property of the history fun[illegible]
purpose is to have status, o[illegible]
course, the memory does h[illegible]
the input history records wi[illegible]
enough information to deter[illegible]

We now show[illegible]
random access memory, wit[illegible]

```
| wait for input

A:      until packet is
        command := p

| analyze input packet

        if command =
            let comma
            send RTR(
        else
```

show that the memory state
the following correspondence

For all X, the contents of me

zero if the inp
Y if the history

Proof by induction on the le
zero by initialization and th

$$\begin{cases} \epsilon \text{ if } x_{N+1} = \text{WRI} \\ \text{RTR(addr, mem} \end{cases}$$

Now $f_{MEM}(X')$

response to X'.

Also, if $x_{N+1}$ =

= $f_{MEM}(X)$ concatenated with

examine the form that the fu

## 2.1.1 FUNCTIONAL SPECIFIC/

A nondetermi

response to a given input hi

multiple valued. One way to

defined by a relation instead

equivalent, is to use function

system $f_{NDMEM}$ that we are d

The rule for realization of a

X with f(X) nonempty, it will

The multiple v

monotonicity property as foll

(3) For all add

The system M

the data in the memory (the

cell) at the instant of the R

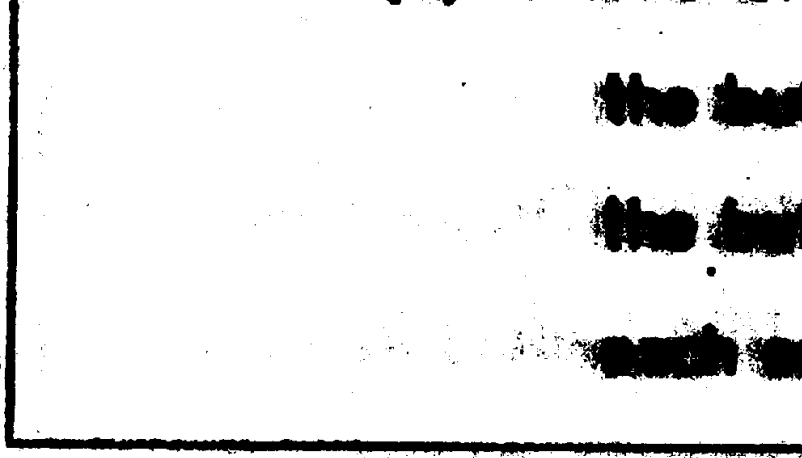
packet is sent out, another W

will have no effect on this R

the bu

the bu

each a



The implement

instantaneous, so it is not ve

a READ packet and the RTR

contents of the memory wor

system to be able to use th

interval. Here is an example

We might like

writes, as well as reads, wha

Sys

(1) When a WRITE pa

on the Pendin

(2) Same as (2) in Sys

(3) Take messages off

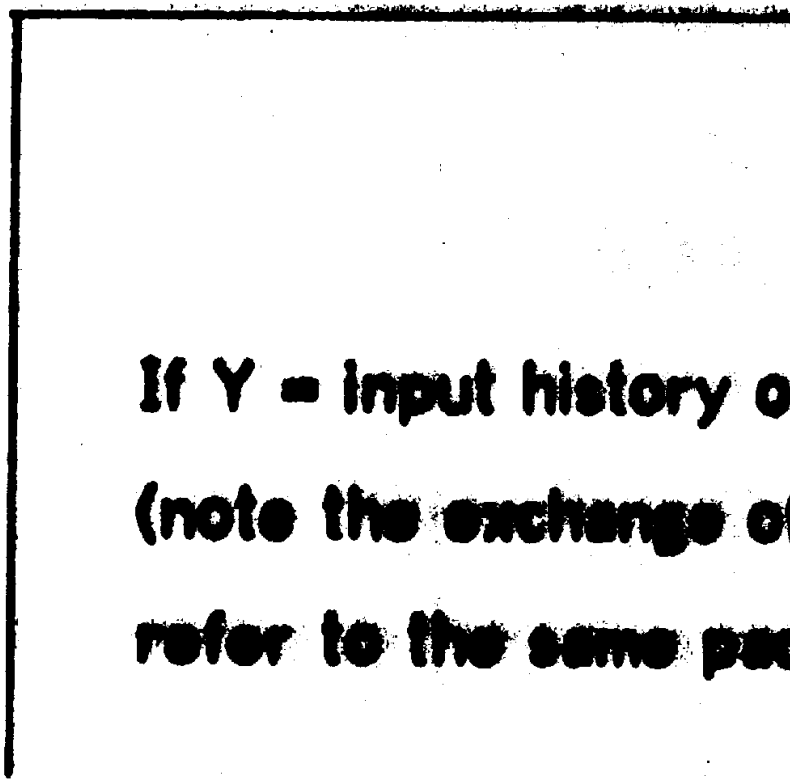
as before, and

(4) Same as (3) in Sys

The requirem

READ/RTR transactions are

nondeterminate functional sp



PWB at $t_0$, the most recent
in the input stream. Therefo

$$D_{addr}(t) = \begin{cases} \text{the data ir} \\ \text{if ther} \\ \text{the conter} \end{cases}$$

we must show that the data
the instant when the READ(
change from $t_0$ to $t_1$. Since
Since the corresponding RTF

This example

Whether or n
given functio
realizes some

There is no w
realize useful functions (eve
that obey certain rules, but

Now suppose
user) violates the rule and
reference count may becom
the count goes to zero, the
nodes.

Two possibili
"spurious" pointer to the cell,
send a READ command to the
that is, that the user has vi

it is only the delay caused
of S. Therefore S's output
time when T was not malfu
both malfunction at precisel
the other at that instant, a
systems conditionally obey
practice.

Until the RTR(

When the RTR(1,data) does

be unable to do anything ab

task of proving that syst

specifications reflect the typ

sense for a system to perfo

input packet not having arri

unreasonable.

indicated by a zero to one
receiver is needed to tell th

The implemen
commands only after the tra
impossible for the user to s
port must also have acknow
be slow and need to be pro
(MEM with acknowledges) mi

(1) If a WRITE packet is rec

one, is

before the i

(3) $|X_A| = |Y_A|$ + number of

(4) $(X_A)_i$ = "ack"

(5) $|Y_A| \leq |M| \leq |Y_A| + 1$

(6) $|M| - 1 \leq |X_A| \leq |M|$

It is easy to

(4) of $f_{AMEM}$. (It is very

Acknowledge Restriction" th

on the user (letting Z = Y).
just exchange "system" and

The S.A.R. is
specification.

In any proof
the weak form of the S.A.R.

We can now
S.A.R. in strong form.

Since the S

acknowledge parts are hand

for the handling of the ack

system. Wherever possible

following way:

It is easy to

operations obeys the Standa

Systems need

example, the implementation

obeyed the Standard Acknow

Here is an imp

identical) to AMEM:

$$(1)\ |Y| = \begin{cases} \text{number of RE} \\ \\ \text{number of RE} \end{cases}$$

$$(3)\ |X_A| = \begin{cases} |X| \text{ if } |X| = 0 \text{ o} \\ \\ |X| - 1 \text{ otherw} \end{cases}$$

This illustrate

quite complicated and requir

A FIFO of lim

following program:

processes start at A,

input port X

output port Y

```
A:      P := RCVPKT()
        XMTPKT(Y) :=
        goto A
```

A FIFO of late

and XMTPKT operations, thou

Appendix I co

N yields a FIFO of length M+

If $X^1, X^2, \ldots$

$(X^1_A, X^2_A, \ldots X^{\ldots}$

(1) $|Y| = \min ($

(2) $\forall i \in [1,N$

(3) $\forall i \in [1,N$

process starts at A

input ports $X_1 \ldots X_n$

output port Y

var p, input

A: wait until a p

| this is nond

input := the p

XMTPKT(Y) :=

input port x

output ports

A:

Higher interc

FIFO buffers.

(2) $|X_A| = m$

(3) $Y^1, Y^2, \ldots$

It may be implemented by

processes start at A

input port X

output ports $Y^1$ ...

queue q size N init

var pop init N

A: wait until a

would be $2^N$ where the addr

Notation: $FET^{(\pm)}$ means any

Each word in

which are used by the str

packets have corresponding

returned unchanged in the c

FET$^{+}$(addr, tag)

FET$^{-}$(addr, tag)

CLR(addr)

is sent. Some of the possibl

$LOAD^{+}(A, D, 2, T1)$ ;

$LOAD^{-}(A, D, 0, T3)$ ;

The reference count tempor

The referenc

in the preceding $LOAD^{(\pm)}$ p

holds in packets. All univ

case obvious from context.

Note: in rul

N+1st DONE in Y, where N

in X. The intention is to let

makes it convenient to thin

or CLR packets.

FET$^{(\pm)}$(addr,--) in X

packets, and D = {

(addr,--,--,--) packe

array [redacted]

var command, addr, [redacted]

A: command := RC[redacted]

if command = [redacted]

let commu[redacted]

XMITPKT([redacted]

else if comma[redacted]

let comma[redacted]

This connecti

increased. Random access

transaction causes the devic

handle any other transaction

during every transaction, a

Putting a FIFO buffer on it

its transaction rate stays t

memory units. If a distribut

an arbitrator can handle

reasonable to use 32 MOS

This connecti

commands at such a slow

situation could arise if sev

rate that the rate of memor

referred to again soon, a
addressed which is not in t
placed in the cache, and al
be cache hits.

The size of t
for the main memory of a c
example, 8 consecutive wo
instruction fetches, tend to
block of 8 consecutive wo
references in the immediate

searched for an item wi
quickly.

A popular
associative" memory [12]
address space is similar
much greater number of
same column in the full
for a given item whose f
If it is in the cache, it m
so only that column of th

row address

data

row address

data

row address

data

The cache h

received requesting the co

The functions

its "top" ports and realize f

P' - Same as P, bu
more FET(±)/LO
The same transa

Q - The first LOAD(
sent as soon a
FET(±)/LOAD(±)
transactions are

intermediate status. If the
either case, some other wor
"modify" flag for that word i

The specified
packets from this host is c
commands have been accepte
allow many commands to be
must accept packets from m
any further commands thro

the item to be.

sending an UPD

at MEMO, wait u

given address, "

fields unspecifie

out through MEM

CLR(addr) at CMDI -

UPD(addr, data, ref)

sending an UPD

STATE P'

FET(±), UPD, or CL

transaction pend

LOAD(±)(addr, data,

into the cache w

memory is accep

cache item to sta

Q'. Q' is simply a t

through MEMO and g

FET(±), UPD, or CLR

commands.

LOAD(±) at MEMI - s

DONE at MEMI - can't

Main memory becom

increment or dec
LOAD⁻ or LOAD⁺.
knows about the
tag) through RES
cache.

DONE(addr) at MEMI

transactions in ma

### 3.2.2 PROOF OF CORRECTNES

A proof of Cl
given in section 2.0.3. The m
last UPD packet in the input
the data in the cache itself
modify bit is off, this data is
in main memory, that is, the
These properties must be s

addressed random access m
speed at any one time. Th
conserve power. The inten
memory. To retrieve any d
directs the CCD to shift that
is happening, the other sec
efficient when many items a
type of packet memory syste

Two other typ
register (it "snakes" back an

the TL at or near the same
they refer. Since the TL is
to many consecutive address

Let «X» be
the corresponding address i
explained later. «X» is jus
memory size to TL size. In
appropriate number of high

Because the

needed. The concepts of "

"counterclockwise from". S

other item, the order of tw

the use of intervals denote

clockwise to Y. If $X \leq Y$, it

from X up to the highest

intervals have their custom

and [Y, X) are clearly compl

The ordering

It is possible

same CCD address. Specifi

One or more FET(±

address, its data

LOAD(±) packets

One or more FET(±)

the appropriate

former case, insert the new
for it by pushing the old ite
the example, insertion of it
Insertion of 42 would requir

While incomi
procedure, packets are bei
through the use of a tran
synchronization with the CC
10, the TLP points to TL add
the CCD pointer = 11, remo

(2) $\forall j \in [ RP , TLP )$,

(That is, cells in the

(3) $\forall j, k \in$ TL address

$j \notin [ \ll a(TL(k)) \gg , k$

(That is, the interval

not contain any emp

(4) $\forall j, k \in$ TL address

then $a(TL(k)) \geq a(TL$

and starts after its end. If
its CCD address is greater
inserted in the first cell and

TL already contains an UP
action, perhaps modifying t

The removal
next to be removed. The C
From the standpoint of the
to be inexorably advancing
clock controlling the shiftin
memory, it is the informatio

The fact that

The above de

minded. Many memory devic

given before the previous

protocol whereby the syste

bearing the data might not

problem can be solved by

one or two bits later, for wr

and may require that the

available. The necessary m

contained in the packet in t
indicated transaction, perh
address of the item which
item is then erased from th
now points to an item havi
same data. All transactions
count changes are noted, an
WRITE packet at CCDO.

When TLP re
address, either it is for a d

After increasing a re
destination until the

For example,
reference count of the resu
destination cells. If one of t
count, the FET$^{+}$ must act fi
arbitrator to be sure the FI
sent until the LOAD$^{+}$ arising
the result to the destination

This can't happen, because
returned.

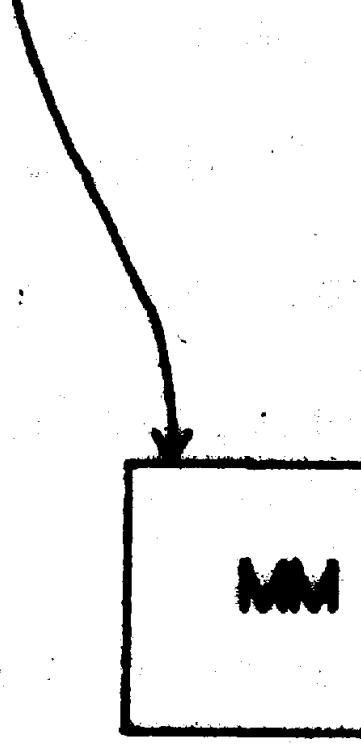
### 5.0.3 MEMORY LATENCY

MM's latency
of MM and its user. Whe
necessary to build a high
performance. For example,
register may be designed t

and thus maintain any desi
controllers are used, each
networks of the data flow
memory transaction rate is

It is not pos
be done with simple funct
module is not correct, bec
memory address space. Th
an interconnection network
direction. Command packe

input port
into tag
(except UPD
packets).




The treatmen
of all pending structure ope
supports a small part of th
part of the total tag space

packet with the same addre
from the two structure con
the FET packet passes thro
rather than the new data
and arbitrator have latency
structure controller receiv
have been accepted by the

If it is not
arbitrators that have no m
passing through the netwo

The procedur

completion of each operati

recursion always terminate

words that need to have t

observed in the case of a l

root node is discarded. All

its count reduced, it goes t

node's offspring.

To implement

(3) Whenever a frod

packet UID(addr,

address of the ru

count of the addr

reference counts can then

instruction cells.

a FET$^{(\pm)}$ packet is receive
FET$^{(\pm)}$ packet is also sent t
memory, the cell in the cac
might seem to make more s
is received from the memor
user - that is, to bypass ste
cache may require writing o
of the LOAD$^{(\pm)}$ packet comin
in response to a packet fro
not accept any replies, the

Why, then, wo

reason is that memory spa

transaction list. The control

level of the memory hierar

expensive) devices, the contr

of the hierarchy, so its trans

In order to us

use "P" cells, the memory sy

"fixed latency law":

share the burden of

In the case

in the cache and memory

transactions in the

another command must go

occur when there are

continuous string of

constant latency, it

in some transactions

controller's user) is needed

will occur.

An efficient s

the needs of programs that

The deadlock

to build a memoryless struct

7. Dennis, J. B., D. P. Misun
Flow Concept. Comput
Science, MIT, Jan. 1977

8. Dennis, J. B. Packet Co
Computer Conference o

9. Keller, R. M. Look-Ahe
177-195.

From the definition of the s