PACKET COMMUNICATION WITHIN
DATA FLOW COMPUTERS

By

JOHN ROBERT KERNS

Bachelor of Arts

Southwest State University

Marshall, Minnesota

1978

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fullfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1982

OKLAHOMA STATE UNIVERSITY LIBRARY

PACKET COMMUNICATION WITHIN

DATA FLOW COMPUTERS

Thesis Approved:

D. A. Thoreson
Thesis Adviser

G. E. Hedrick

Donald D. Fisher

Norman N. Durham
Dean of the Graduate College

1131321

PREFACE

This study examines the operation and organization of three data flow computer systems, the systems from MIT, the University of Manchester, and Texas Instruments Inc. Each of the three systems is described in an idealized configuration; then the idealized systems are analyzed. Expected packet throughput within different modules of each system, and of the systems themselves, is derived. The maximum throughput expected of the idealized systems are compared with present day systems, in order to check the validity of the throughput analysis. In addition, the advantages and disadvantages of each of the three data flow systems are discussed.

I would like to express my gratitude to my major adviser, Dr. Sharilyn A. Thoreson, for her valuable guidance and advice throughout my thesis research. I also acknowledge my other committee members, Dr. George E. Hedrick and Dr. Donald D. Fisher, for their many helpful suggestions. In addition, I would like to express my appreciation to the other professors and graduate students I have worked with, for their many discussions on computer science, encouragement, and friendship throughout my graduate studies.

This document was prepared and edited using SCRIPT Version 3.6, a text formatting system from the University of Waterloo. An IBM 6670 laser printer was used during the development of the document and was used for the final printing of the thesis.

I would like to thank both the Department of Computing and Information Sciences and the Center for Applications of Remote Sensing for their financial support during my graduate studies.

Finally, I would like to express my appreciation for the support and understanding of my aunt, Matilda Wulf, and my mother, Lois Kerns.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

Project Motivation 4
Data Flow Computation Theory 5

II. DATA FLOW COMPUTER SYSTEMS 11

MIT Data Flow Computer 11
Manchester Data Flow Computer 14
Texas Instruments Data Flow Computer . . 19

III. IDEALIZED DATA FLOW COMPUTER SYSTEMS 26

Components
Processing Elements 28
Memory 30
Control Units 35
Arbitration and
Distribution Units 37
Idealized System Configurations
MIT Idealized System 40
Manchester Idealized System 46
Texas Instruments Idealized System . 53

IV. ANALYSIS OF IDEALIZED DATA FLOW SYSTEMS . . . 60

Packet Rates Through
Interconnection Network Modules
MIT Idealized System Rates 62
Manchester Idealized System Rates . 64
Texas Instruments
Idealized System Rates 65
Analysis of Packet Rates 68
Advantages and Disadvantages
of Selected Data Flow Systems 73
Real Machine Comparison 74

V. SUMMARY, CONCLUSIONS,
AND SUGGESTED FUTURE RESEARCH 77

Summary and Conclusions 77
Suggested Future Research 80

SELECTED BIBLIOGRAPHY 81

Chapter Page

APPENDIX A - PMS DESCRIPTIONS OF THREE IDEALIZED DATA FLOW COMPUTER SYSTEMS 86

APPENDIX B - TIMING FOR SINGLE PACKETS TO PASS THROUGH INTERCONNECTION MODULES . . . 92

APPENDIX C - LIST OF SYMBOLS 96

APPENDIX D - LIST OF EQUATIONS 99

APPENDIX E - GLOSSARY 103

LIST OF TABLES

Table	Page
I. Ratio of Memory Access Time to Addition Time in Actual Machine Cycles . . .	32
II. Memory Access Cycles for Serial Access and Interleaved Memory Access	33
III. MIT Idealized Module Timing and Rate Figures .	67
IV. Manchester Idealized Module Timing and Rate Figures	67
V. Texas Instruments Idealized Module Timing and Rate Figures	68
VI. MIT Idealized System Module and Component Timing	93
VII. Manchester Idealized System Module and Component Timing	94
VIII. Texas Instruments Idealized System Module and Component Timing	95

LIST OF FIGURES

Figure		Page
1.	Sample Program on a von-Neuman Computer . . .	7
2.	Sample Program in Data Flow Graph Representation	8
3.	MIT Basic Data Flow Processor	12
4.	Manchester Data Flow Processor	16
5.	Matching Store Cycle	17
6.	Instruction Store Cycle	18
7.	Texas Instruments Data Flow Processor	21
8.	Single Processing Element Network	22
9.	Single Operation Unit Connected to Ring . . .	23
10.	Idealized System Components	27
11.	Arbitration and Distribution Units	39
12.	MIT Idealized System	41
13.	MIT Packet Formats	42
14.	MIT Interleaved Memory	44
15.	Individual Cell in Cell Memory	44
16.	Manchester Idealized System	47
18.	Manchester Packet Formats	49
18.	Manchester Interleaved Memory	49
19.	Texas Instruments Idealized System	54
20.	Texas Instruments Packet Formats	56
21.	Texas Instruments Interleaved Memory	56

Figure | Page

22. MIT Idealized Data Flow System 88

23. Manchester Idealized Data Flow System 89

24. Texas Instruments Idealized Data Flow System 90

25. Texas Instruments Single Operational Unit . . 91

CHAPTER I

INTRODUCTION

Ever since the first computers were built in the late 1940's and ealy 1950's, there has been a constant push to build computer systems with more computing power and greater throughput. In the early years of computer science, the continued advance of new technology gave rise to larger and faster computer systems. Vaccum tubes were replaced by single transistors, which were incorporated into integrated circuits. Of course, these new technologies greatly improved processor and memory speed and greatly increased throughput over earlier systems.

However, users never are satisfied completely as newer and more complex applications are developed for computer systems, and larger and faster computers will be needed to run those applications. Some limits exist in the development of new hardware technology and are already being approached in the laboratory. Because of technology limitations, advances in circuit technology must be aided by advances in computer architecture and software in the evolution of new computer systems.

Problems such as weather forcasting, nuclear reactor simulation, and three dimensional fluid flow analysis, continue to tax the largest and most powerful systems built to date. The characteristics of these and similiar problems spawned new architectures for several new systems. Programs using large vectors and arrays can take advantage of parallel processors such as the ILLIAC IV [9] and vector processors like the CDC Star 100 [29] and the Cray 1 [44]. Two techniques used in supercomputers to help achieve high throughput are: the use of parallel processors and pipelining.

Parallel processing is a technique which utilizes a set of processors that operate in parallel; programs using large vectors can take advantage of parallel processors by letting each processor operate on a different element of the vector.

In a pipelined processor, the processor functions are distributed over several modules; the total work rate is increased by overlapping execution of the modules. The Cray 1 uses piplined operation units to help achieve its throughput rate of 125 - 250 million instructions per second (MIPS); this makes it one of the fastest and most powerful computers built.

Unfortunately, only a small class of real world applications can take advantage of processors such as these. Vector and parallel processors need applications using large vectors or arrays to fully utilize the

processing power available. Programs that are highly sequential or dominated by input and output cannot utilize vector and parallel processors effectively.

One approach to building larger and faster systems is to connect many processors together and let each processor work on a share of the work load. Computer systems using several tightly-coupled processors can usually increase system throughput by adding more processors to the system. A tightly-coupled computer system has many computers, where each computer depends on the others for continued operation. Distributed computer architectures are usually one of two classes: a single instruction multiple data stream system (SIMD), or a multiple instruction multiple data stream system (MIMD). The ILLIAC IV is an example of a SIMD architecture. A data flow computer is an example of a MIMD architecture.

In a data flow processing element, typically more than one processing element is connected in a multiple processor system, where instructions can execute on any of the available processors. The processors are arranged so each processing element can operate concurrently. Programs run on a data flow computer have a machine representation which takes advantage of processors that execute in parallel.

Processors require an interconnection network for communication of instructions and data between different modules of the system. Interconnection networks should

not be confused with local networks like ETHERNET where external devices are connected by a network, or with external networks like ARPANET where separate computer systems have communication links with other computers in the network. Interconnection networks consist of the components of the system (processors, memories, control units, and switches) and the communication lines among them.

Some multiple processor systems use a method of information transfer called packet communication. A set of information is placed together in a packet and exchanged among processors. Contents of a packet include instructions, data, and control information. Packet sizes vary among different systems, from several bytes up to several thousand bytes. In multiple processor systems using packet communication, the processors do not need to monitor the activity in the other processors. As long as packets can be received, the information in the packet processed, and results sent to appropriate destinations, each processor can operate independently.

Project Motivation

Until recently, all computers used very basic architectural principles proposed by John von Neumann in 1945 [47]. Two main principles of such systems are stored programs and sequential order of program execution. Programs and data are stored in main memory, and a

sequential control is used to govern the operation of the machine. Data flow architecture deviates from the von Neumann principles. Data flow computers operate asynchronously without sequential control and use a distributed memory instead of a single updatable memory.

Research in the area of data flow computers is still very new, and only a few operational data flow computers have been built. At this time, it is not obvious which of the new data flow architectures will be successful, or how they will compare. The relative performance of different systems can be analyzed without actually building prototype systems. This is an analysis of the relative performance of three data flow systems, achieved by computing and comparing the maximum throughput rate and minimum execution time.

Data Flow Computation Theory

Programs run on a data flow computer are a partially ordered set of operations or instructions. The machine representation of a data flow program is a directed graph; where nodes represent instructions and arcs between nodes represent data dependencies between instructions. An instruction is ready to execute only when, and as soon as, all operands have been received.

There are two types of data flow architectures, data driven and demand driven [48]. Data driven architectures are studied here. In data driven computers, the

availability of all operands for an instruction triggers execution of that instruction. In demand driven computers, the requirement for a result triggers the instruction that will generate that result.

The data flow graph of a program deals only with data values and not with data addresses in contrast with most high level languages. As a result, there is no need for a single updatable memory; instructions produce results, which are in turn used by other instructions. Also, there is no need for a single controller to control instruction execution.

Following are two expressions to be computed on both a von Neumann computer and on a data driven data flow computer. The following expressions come from the quadratic formula:

$$X1 = \frac{- B + \sqrt{B^2 - 4*A*C}}{2*A}$$

$$X2 = \frac{- B - \sqrt{B^2 - 4*A*C}}{2*A}$$

Breaking the expressions into simple instructions, the program in Figure 1 will execute on a von Neumann machine in 12 operations. Figure 2 shows the same program represented as a data flow graph. Operations that can be done in parallel are placed at the same level in the graph. Because the longest path in the graph includes

```
Input A, B, C
A2   =  A * 2
A4   =  A * 4
NEGB = - B
BSQ   =  B * B
A4C   =  A4 * C
BSQMA4C  = BSQ - A4C
TMPSQRT  =  Sqrt (BSQMA4C)
TMPX1  =  NEGB + TMPSQRT
TMPX2  =  NEGB - TMPSQRT
X1  =  TMPX1 / A2
X2  =  TMPX2 / A2
```

Figure 1. Sample Program on a von Neumann Computer

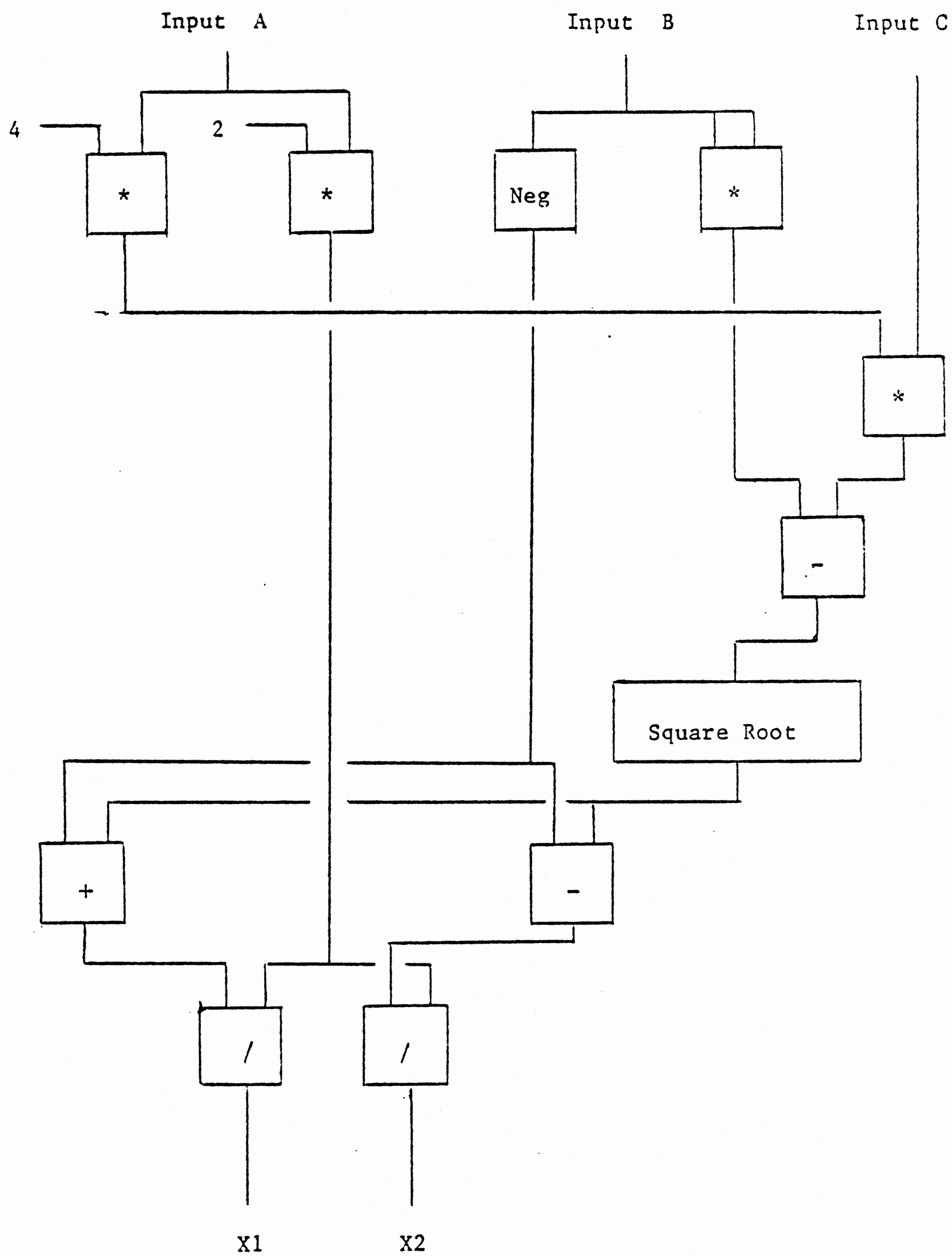


Figure 2. Sample Program in Data Flow Graph Representation

seven instructions, the program can execute in the time required to perform seven instructions, if enough processors (4) are available. The amount of speedup gained by parallel processors is limited by the amount of parallelism in the data flow graph. Minimum execution time of a program is determined by the longest path from the beginning to the end of the data flow graph.

There are several modules needed in a data flow computer. Processors are arranged to operate in parallel. A memory is needed to hold instructions ready to execute, and a memory is needed to hold instructions waiting for operands. Routing units are needed to direct the flow of packets, and control units are needed in memory modules. Communication lines between modules are also required. Where possible, parallelism within a module is exploited as much as possible.

The architectures of several data flow computers were described in recent literature [48]. Two architectures have been developed at the University of Utah [14, 15, 33]. A research group in Europe called CERN has built a prototype data flow computer as described in [42]. And a data flow system was being designed at the University of California at Irvine [4, 5, 8]. Three other architectures are described in this thesis.

Chapter II contains a description of the architectures and interconnection networks of three data flow computer systems. All three systems use the data

driven approach. The three systems were chosen because of their unique data flow architectures. In the Texas Instruments design, each processor has a separate memory unit. In the MIT design, all processors have access to a parallel memory. In the Manchester design, several memories are used in the matching of operands before instruction execution.

Chapter III gives a description of the components used in the idealized system configurations for comparison. Each component is described, and its timing equations are given. Using component timing, the timing for each idealized module is derived for each system.

In Chapter IV, maximum rates of packet throughput are derived for modules of the three idealized systems. The rates are used for an analysis of the minimum and maximum rate of the different systems. Advantages and disadvantages of the three systems are discussed also. The maximum rate figures are also compared with a present day supercomputer, the CRAY 1.

Chapter V states the summary and conclusions of this research. Suggestions for future research are also given.

CHAPTER II

DATA FLOW COMPUTER SYSTEMS

MIT Data Flow Computer

The data flow system developed at MIT by Jack Dennis has been developed in stages [22, 23]. The stage presented here is one of the earlier designs called the "Basic Data Flow Processor" [23], (Figure 3). This stage was chosen because it has features similar to the other two systems being described in this thesis. Later stages of the Dennis design include features not found on the other two systems, making the later revisions inappropriate for this comparision.

There are four major modules arranged on one ring: processing elements, a memory for instructions called a cell memory, an arbitration unit, and a distribution unit. The distribution unit and arbitration unit act as routing directors for the packets in the ring.

Instructions waiting for operands are held in the cell memory. Each instruction is released or enabled for execution when all of its operands have been received. Computation of results is done in the processing elements as they receive instructions.

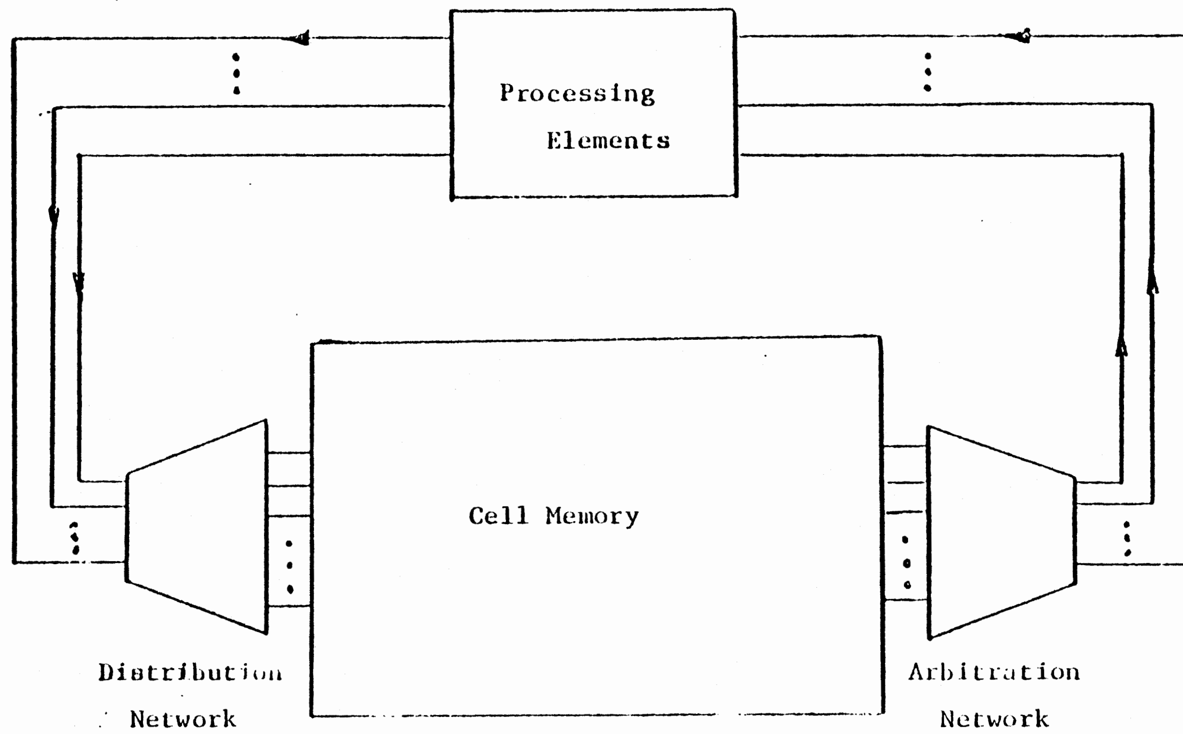


Figure 3. MIT Basic Data Flow Processor

Having one or more input lines and one or more output lines, a routing unit accepts a packet and routes it to the correct module. An arbitration unit is a routing unit with more input lines than output lines. A distribution unit is a routing unit with more output lines than input lines or with the same number of input and output lines.

The architecture of the MIT design allows each module of the system to operate independently of the other modules.

The cell memory is divided into a number of cells, each cell large enough to hold many instructions. Each cell in the cell memory contains a separate control unit attached to a memory. The cell memory is multiported to allow access to each cell separately. This allows concurrent operation of the cells in the cell memory. There are several different instruction formats for different types of instructions, Dennis [19] has a complete description. When all operands for an instruction are received, an instruction packet is formed, enabled, and sent into the arbitration unit. Packets released from the cell memory contain the operation code, two destination addresses for the storage of the result to be generated, two operands, and any other needed control information.

When a processing element becomes idle, the arbitration network sends an instruction packet to that processor, and the instruction is executed. Instructions

produce results which are placed into result packets; result packets released from the processing elements contain a result operand and the destination of that operand. As results from the processing elements are generated, they are routed into the distribution unit. The distribution network then routes the packet to the correct cell in the cell memory.

The following actions are typical of packets travelling around the interconnection network. After result packets leave a processor, they enter the distribution network where the destination address is used to direct the result packet to a specific cell in the cell memory. The operand is stored directly in the instruction packet, the destination address is then discarded. When the control unit associated with each cell detects that all operands have been received, the instruction is enabled and an instruction packet is sent into the arbitration unit to the next available processor.

Manchester Data Flow Computer

A detailed description of the Manchester data flow computer can be found in [28, 49, 50]. In the Manchester design there are five major modules arranged around a ring as shown in Figure 4. Two features of this architecture, which are unique from the other two architectures, are the matching store and instruction store. The matching store holds an instruction's operands until all of them have

arrived. The instruction store holds operation codes and destinations for instructions ready to execute when the operands arrive.

The matching store is controlled by the matching store cycle, as shown in Figure 5. Result packets are accepted from the result queue, the matching store is checked for a matching operand, and matched operands are sent on to the instruction memory. The matching store acts as an associative memory by hashing the address, using a name composed of four parts (activation name, iteration level, index number, and instruction number) to find the correct instruction.

The instruction store is controlled by the operation of the instruction store cycle shown in Figure 6. Instruction packets are formed from the operands and the instruction in the instruction store. Instruction packets are sent to processing elements as they become available.

The processing elements must accept and unpack instruction packets, execute the instruction, and form result packets. As result packets are generated, they are forwarded to the I/O switch and then to the result queue.

The result queue is a buffer to hold result packets until the matching store can process them. Because of the arrangement of processors, the result queue must be able to handle bursts of result packets. Packets are placed at the end of a queue. When the packet advances to the front of the queue, it can be released to the matching store.

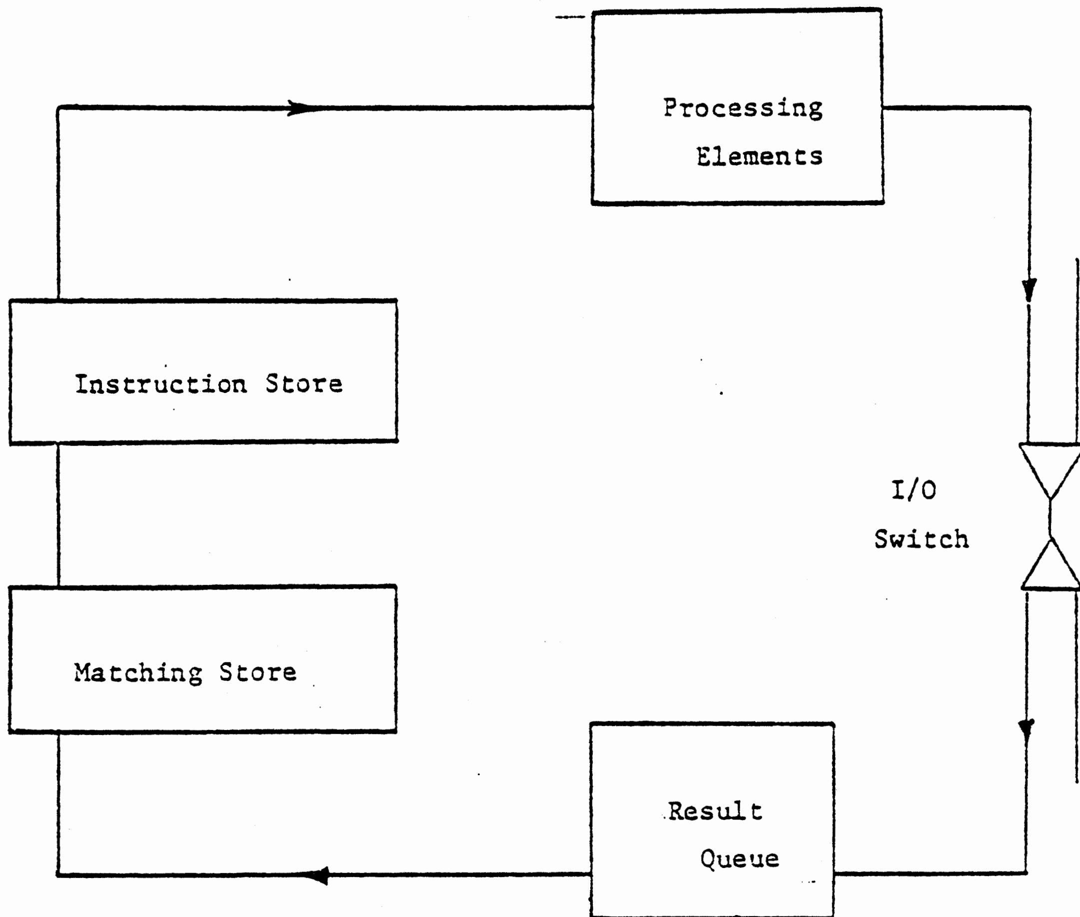


Figure 4. Manchester Data Flow Processor

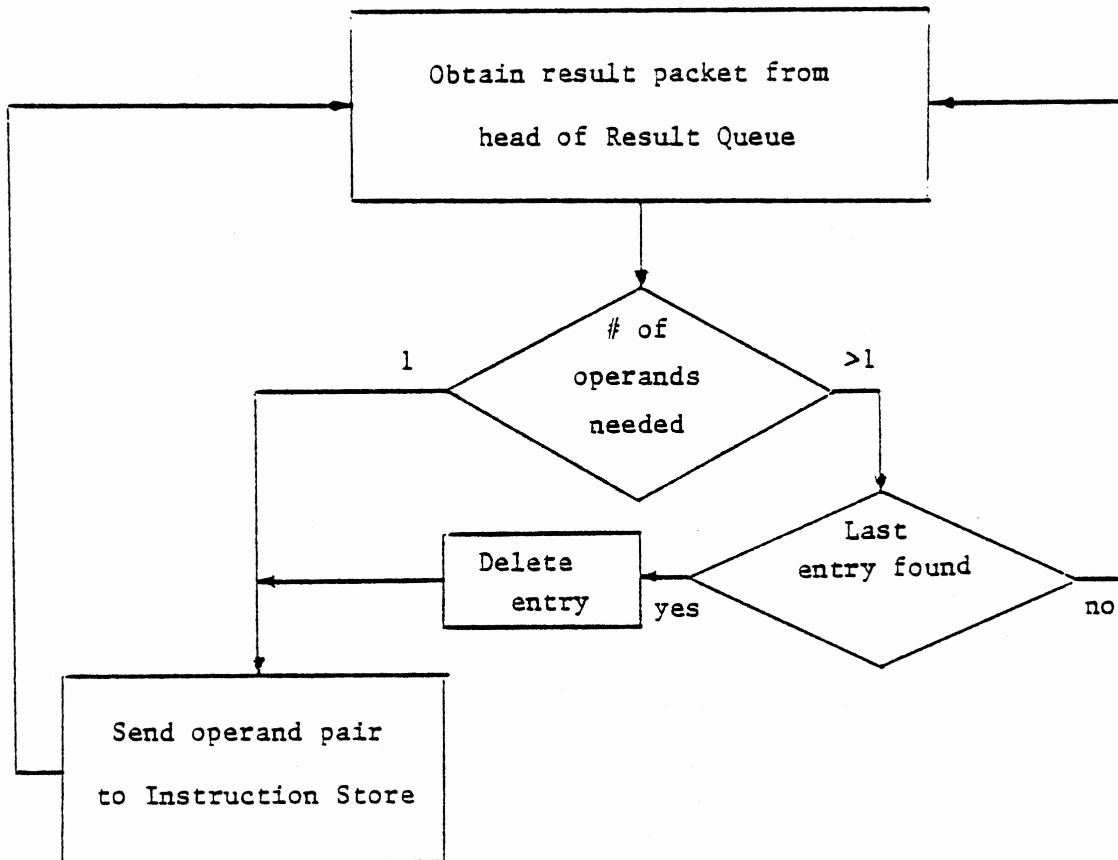


Figure 5. Matching Store Cycle

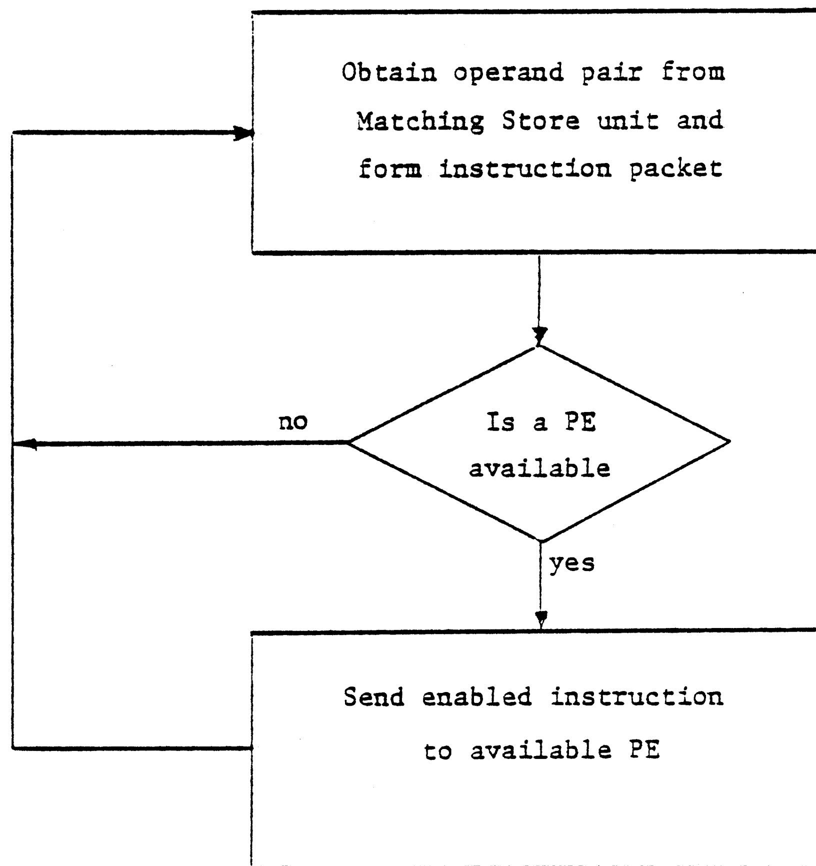


Figure 6. Instruction Store Cycle

A typical circuit of the Manchester interconnection network proceeds as follows. As a result packet leaves a processor and the PE (processing element) arbitration unit, the packet passes through the I/O switch and is stored at the end of the result queue. When the packet arrives at the front of the queue, it is forwarded to the matching store unit. In the matching store, a search is made for an operand with the same destination address. If no operand is found for a binary instruction, the operand will be stored until the matching operand arrives. Once operands are matched, the operand pair is forwarded to the instruction store; results for unary instructions bypass the matching store and pass directly to the instruction store. The operand pair is held in an internal buffer in the instruction store control unit while the instruction is retrieved from the instruction store memory. The instruction and operands are then combined into an enabled instruction packet and sent to the first available processing element.

Texas Instruments Data Flow Computer

Information about the Texas Instruments (TI) data flow design is less complete than others because it comes from an industrial laboratory. The TI Distributed Data Processor (DDP) was one of the first data flow computers built [31, 45]. TI's data flow computer was built as a laboratory demonstrator to show the validity of data flow

computation principles. It is unique in using FORTRAN as its source language.

The TI data flow computer is composed of four operational units connected to a circular interconnection network, as shown in Figure 7. Each operational unit acts as a separate data flow processor with a processing element, a local memory, and a separate interconnection network as in Figure 8. As expected, each operational unit operates independently, and each PE operates in parallel with the others.

In attaching the operational unit to the ring, an instruction/result port is added to the ring interface, and buffers are added to the processing element and memory unit to store incoming packets until they can be used, see Figure 9. As a packet goes through the instruction port, the PE number is checked, and the operand count is checked to determine if the packet goes to the processing element or instruction memory. In a similar manner, as packets leave the processing element, the PE number is checked; result packets are sent either onto the main ring or into a buffer.

The purpose of buffers is to hold packets until they can be processed by the PE or the instruction memory. Incoming packets are placed at the end of a queue; when the packet arrives at the front of the queue it is released when the PE or memory becomes idle. The buffers can both accept incoming packets and process outgoing packets at the same time.

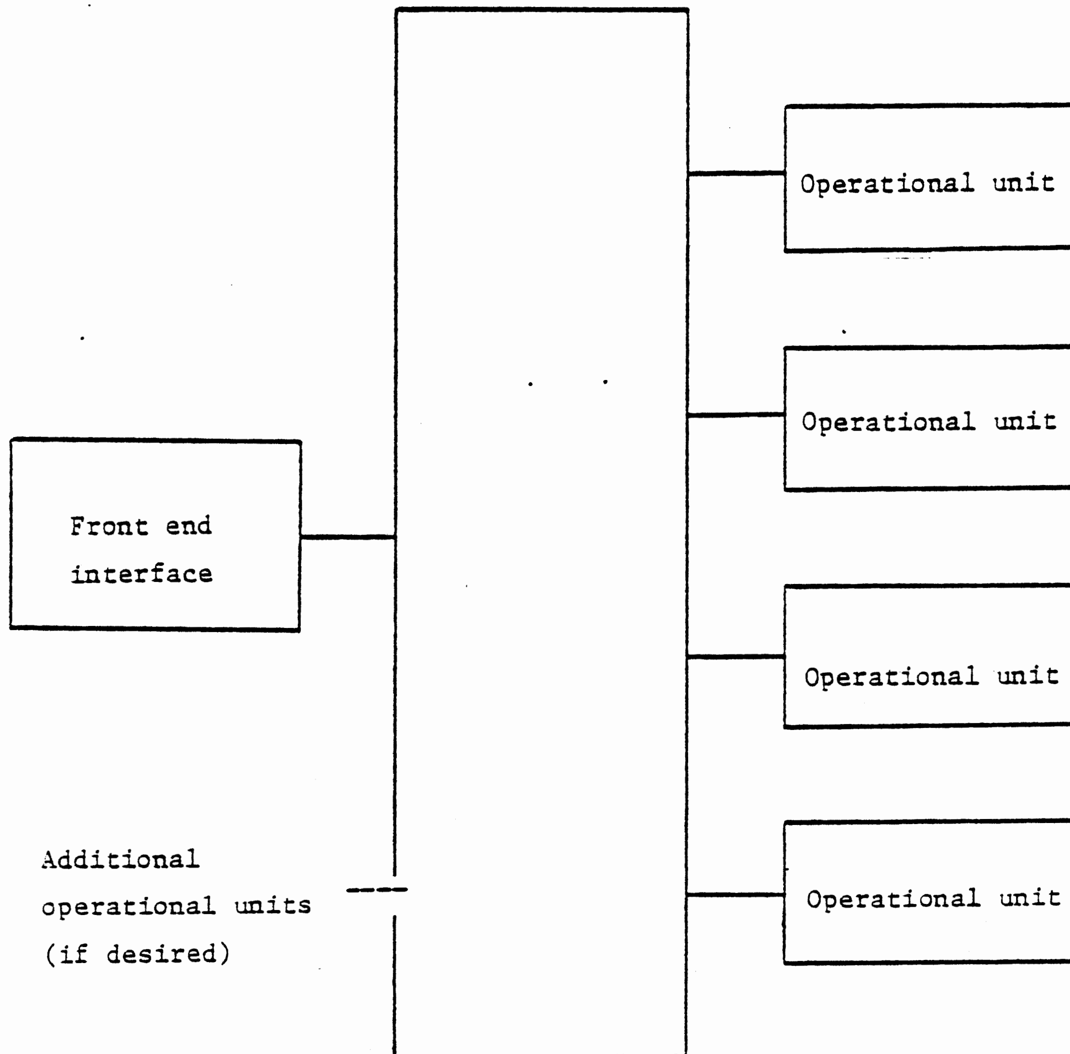


Figure 7. Texas Instruments Data Flow Processor

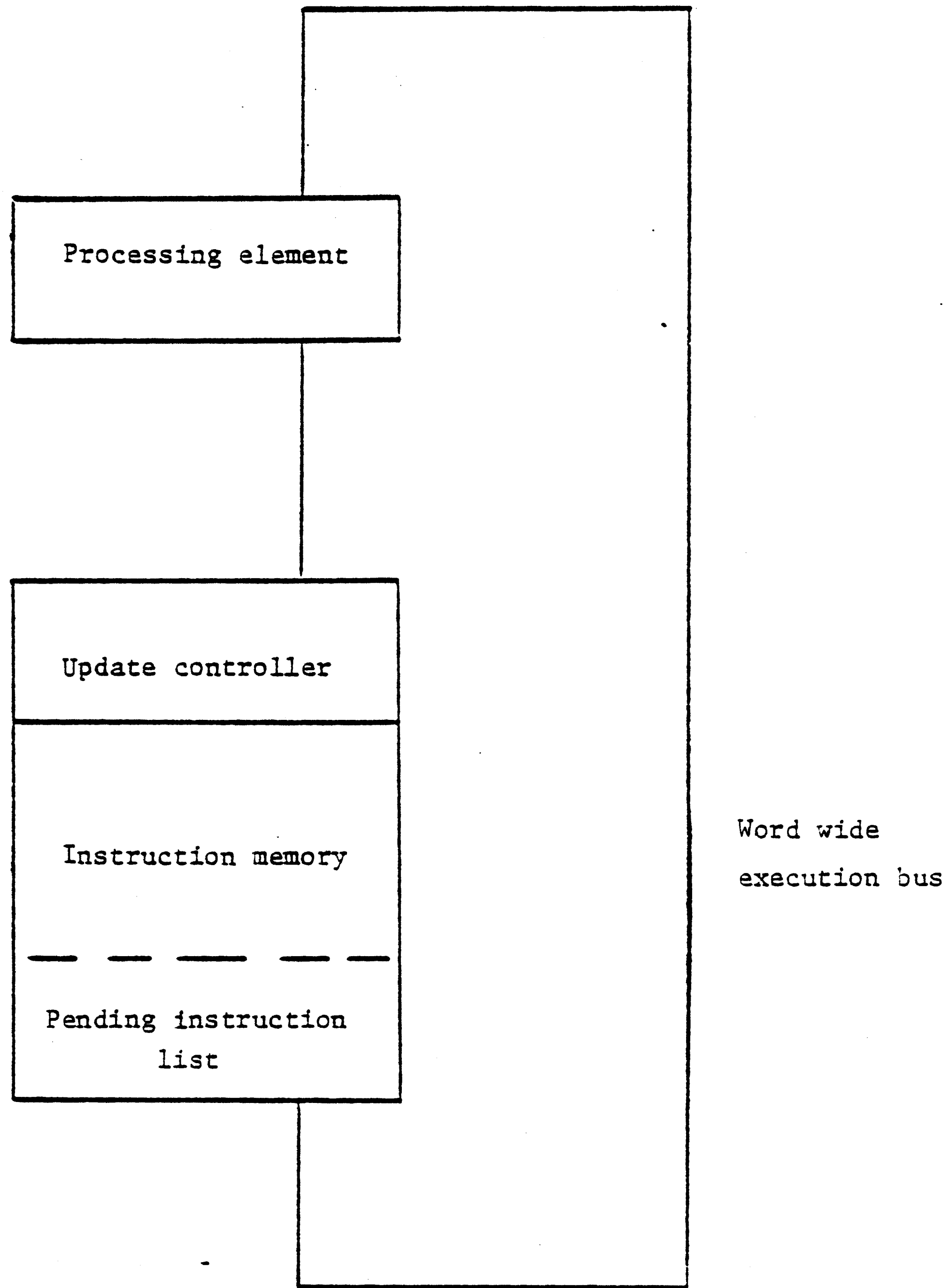


Figure 8. Single Processing Element Network

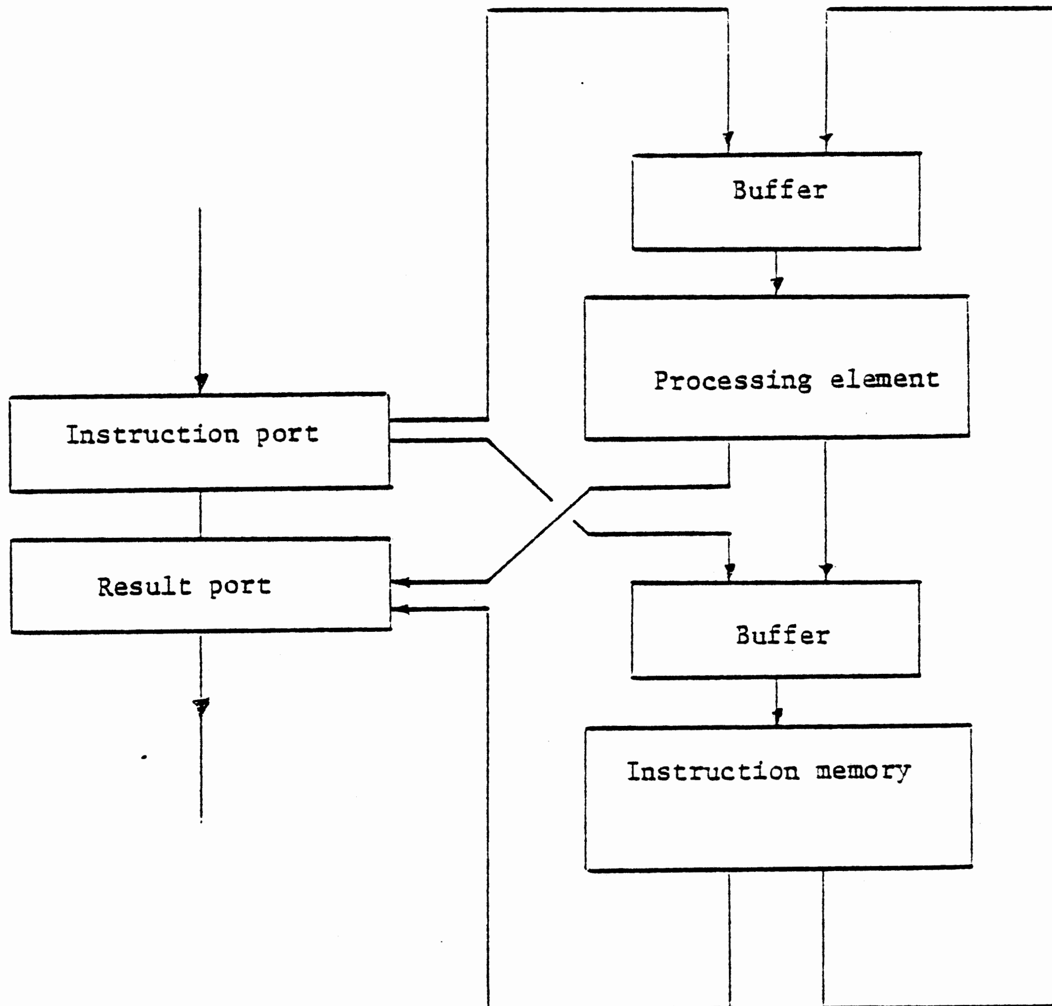


Figure 9. Single Operational Unit Connected to Ring

In the instruction memory, an operand is placed in the instruction, and the instruction is enabled if all operands are in the instruction. The instruction memory needs a control unit along with memory units. When an instruction is enabled, an instruction packet is formed and forwarded.

A typical circuit of a packet around the operational unit would proceed as follows. As result packets leave the processing element, a distribution unit checks if the packet should stay in the same operational unit or be sent onto the main ring to go to another operational unit. Result packets are sent to the memory buffer unit where they are placed at the end of a queue. When the packet arrives at the front of the buffer, it must wait for the memory unit to be idle before it can proceed. Once a result packet is sent to the instruction memory, the instruction is read into the control unit from memory. The number of operands needed for the instruction is decremented and tested. If zero, the instruction is enabled, and an instruction packet is formed and sent on. Otherwise, the instruction is stored back into memory.

Depending on the destinations in the instruction packet, the packet could be sent out of the operational unit or back to the PE buffer. The buffer in front of the processing element acts as a queue, like the buffer in front of the instruction memory. When the packet works to the front of the queue, it is passed on to an available processing element.

One difference in the TI design is that the data flow graph of the program must be partitioned in subgraphs to fit into the different operational units. Because of expected localized activity in the operational units, there is not expected to be a lot of traffic on the main ring. However, a PE number is a part of all destination addresses to aid in the correct routing of packets.

The interconnection network is a packet wide ring, 16 words long, passing by the four processings elements and the front end interface. Instruction and result packets are placed on the ring, logically expanding or collapsing the size of the ring as required. The instruction/result port removes packets and places new ones on the ring as required. This allows a variable number of packets to circulate around the interconnection network.

From the descriptions given in this chapter, it can be seen that each data flow system uses similar components. These components are processors, memories, routing units, and control units. However, the arrangement and use of the different components between the three systems shows many differences. The differences in the systems will be examined after the similar components are analyzed.

CHAPTER III

IDEALIZED DATA FLOW COMPUTER SYSTEMS

In order to compare the interconnection networks of the three data flow computers, the specific computers analyzed should have configurations with similar capacities. The major modules of each system are examined separately to determine the configurations.

Each system is composed of several modules connected to an interconnection network. A module is a major unit in the system and is composed of one or more components. A component is a single device or a set of similar devices. Four components used to describe the actions of each module are: memory, processing elements, control units, and arbitration and distribution units. Each data flow system is described using these components; each system configuration is defined so each system has similar capacities. Each idealized system has four processing elements, and 512K bytes of memory; each instruction packet has space for two operands and two destination addresses. The components will be represented by the symbols in Figure 10.

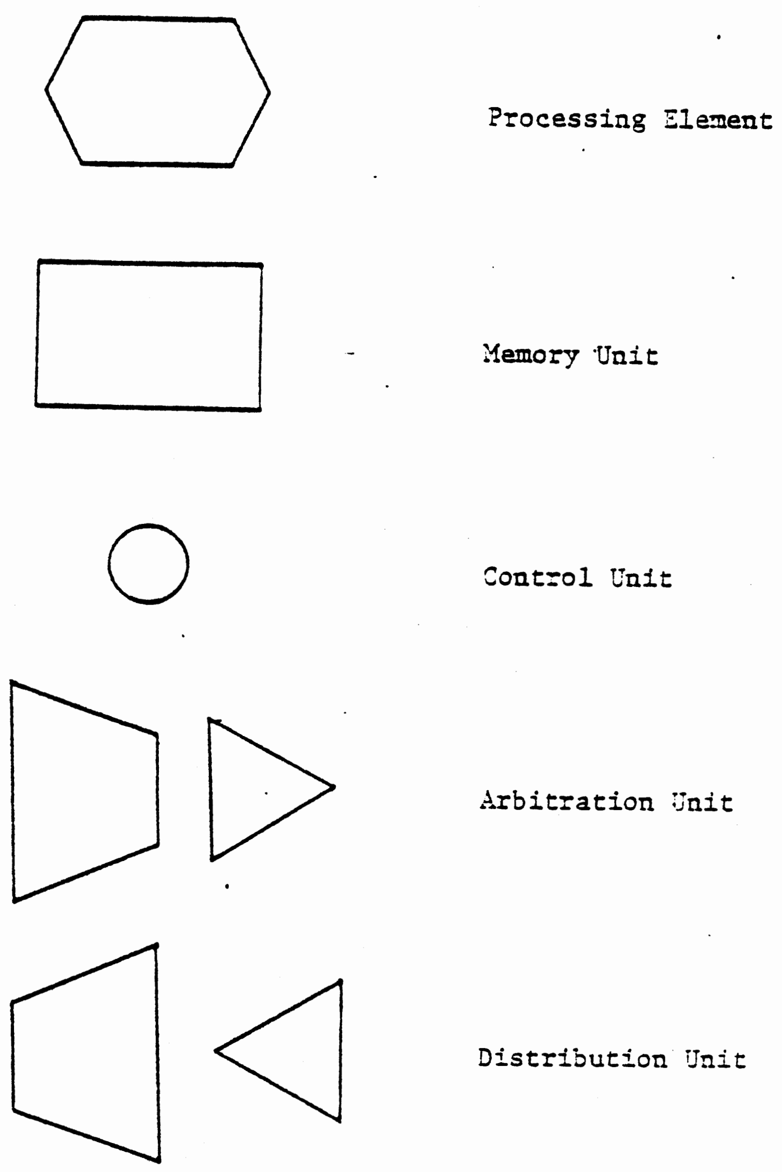


Figure 10. Idealized System Components

In order to avoid technology - dependent timings, timing is expressed in number of gate delays through a particular module. A gate delay is the time for a unit of information to pass through a simple electronic device or gate. Equations to describe the rates for each module together with specific values for the idealized system being analyzed are given later in this chapter . The form used is:

T.x = General timing equation for x (Eq#)
= Value for specific idealized system.

All timing equations are listed together in Appendix D.

Specific timing for each module is dependent on component timing. Timing for a single packet to pass through a module assumes both that no other packets are active and that the module is idle.

Components

Processing Elements

The processing elements used in the data flow systems described here support arithmetic and boolean operations. A processing element also can support operations like SINE, COSINE, LOG, and EXP. This analysis uses the time for one addition as the processing element timing. Instruction packets that are sent to the PE contain one or two operands, an operation code, and destination addresses

for the results. The processors used in a data flow computer must unpack the instruction packet, decode the operation code, process the instruction, and form result packets. There is no need for the processing element to fetch operands from memory or to store results in memory.

Operands are all 32-bit words with possible data types of integer, real, or boolean. Because the processing elements are designed to execute only single instructions, there is no need to direct results or incoming instructions. Result/destination pairs are sent out immediately, and a new instruction is forwarded to the PE as soon as it can be made available.

The time for an instruction to pass through a PE is defined as the number of gate delays in doing an addition on two full words. An analysis of gate delays in arithmetic expressions is given in [34], where the time to perform an addition is given as:

$$T.add \leq 1/2 * (5 + \log_f t) * \log_2 t + 4. \qquad (3.1)$$

In this equation, f is the fan-out of the gates, and t is the size of a full word in bits. Using a fan-out of 8 on 32-bit words, the number of gate delays to do an addition is computed as:

$$\begin{aligned} T.add &\leq 1/2 * (5 + \log_8 32) * \log_2 32 + 4 \\ &\leq 21. \end{aligned}$$

Since timings are approximate, 20 is used as the minimum time for an instruction to pass through a processing element. This timing does not include time to unpack the instruction packet, decode the operation code, or to form result packets. Each result packet sent out from the processing elements contains a single result and a single destination address. If two destination addresses are in the instruction packet, two result packets will be sent out from the PE.

Memory

Because a memory device does not operate using combinatorial logic, the number of gate delays in a memory is difficult to estimate. Actual instruction timing can be used to approximate the time through a memory in terms of gate delays[1]. By using the number of machine cycles required to do an addition and the number of machine cycles to do a memory access, a ratio of memory access machine cycles to addition machine cycles can be used to approximate the number of gate delays through a memory. Using the number of gate delays required for an addition, the number of gate delays required for a memory access can be calculated using the Memory - Addition Ratio. Table I shows the ratio of memory access time to addition time in machine cycles for several micro-computers. In

---

[1] Personal communication with Dr. Louis Johnson, Dept. of Electrical Engineering, Oklahoma State University, (1982).

calculating the ratios, register to register addition was used for the addition time, and register to memory time was used for a memory access time [41]. The memory - addition ratio for each machine is given as:

Memory - Addition ratio =
((number of cycles to read + number of cycles to write)/2)
/ (number of cycles to add)

The average ratio for the memory access to an addition is calculated to be 1.95; however 2.0 is used for convenience since this is an approximation. The time for a memory access will then be defined as

$$T.mu = T.pe * 2 = 40. \tag{3.2}$$

The use of packets in an interconnection network requires that the bandwidth of the network communication lines and the size of the packet be the same. The bandwidth is the number of bits of information that can be sent over communication lines. When storing or retrieving a packet from memory, the organization of the memory greatly affects the total time of packet transfer. If the memory unit is only capable of accessing one byte or one word at a time, then accessing a packet of information in memory would require more than a single read or write operation. This method of memory access would be inadequate for large packets because of the reduction in memory access speed.

TABLE I

RATIO OF MEMORY ACCESS TIME TO ADDITION TIME
IN ACTUAL MACHINE CYCLES

Microprocessor	Add	Store	Load	Ratio
Fairchild F8	1.0	2.5	4.0	3.25
Intel 8080A/8085	4	7	7	1.75
Intel 8048	1	2	2	2.00
IM 6100	1.0	1.7	1.7	1.70
Gen. Inst. CP1600	8	11	10	1.31
Micronova	5	6	6	1.20
Motorola MC 6800	2	6	5	2.75
Motorola MCS 6500	2	6	6	3.00
Nat. Semi. PACE	4	4	4	1.00
Nat. Semi. SC/MP	7	18	18	2.57
RCA CDP 1802	2	2	2	1.00
Signetics 2650	2	3	3	1.50
SMS 300	1	1	1	1.00
TI TMS 9900	14	60	52	4.00
Z80A	7	9	9	1.29

Average ratio of Memory access to Addition = 1.95.

To avoid this reduction in memory access speed, a packet of information should be read or written during a single operation. To do this, the memory can be separated into several banks of storage operating in parallel. This method of memory organization is called interleaving and commonly is used to achieve a high bandwidth during memory access. If a packet size is four words long, the memory can be divided into four separate banks of storage; a read or write can then be done in one memory access time by simultaneously accessing each memory bank. Table II shows

the difference in time between using serial memory access and interleaved memory access. In the descriptions of the idealized systems, interleaved memory structures are used.

TABLE II

MEMORY ACCESS CYCLES FOR SERIAL ACCESS AND INTERLEAVED MEMORY ACCESS

Processor	Packet Type	Serial Access (gate delays)	Interleaved Access (gate delays)	
MIT				
	Result	80	40	2-way
	Instruction	160	40	4-way
Manchester				
	Result	120	40	3-way
	Instruction	80	40	2-way
Texas Instruments				
	Result	80	40	2-way
	Instruction	160	40	2-way

Many memory units are said to have nondestructive readout, that is, a read operation does not destroy the contents of the memory location. This usually involves the use of an intermediate buffer in the memory where the contents of the location just read are placed. This buffer is then written back into the same location to insure that the contents are not destroyed. It is desirable to have memory with destructive readout capabilities in a queue.

capabilities in a queue. Destructive readout causes the contents of a specified memory location to be destroyed on a read operation. It is assumed without loss of generality that zeroes replace the previous contents of that location after a read operation.

Some memory units have an additional feature that can be used in data flow computers. A memory normally has a single port for both input and output, but a multiported memory allows more than one operation in a memory unit at the same time. A single-ported memory only has a single access path to read/write data from memory. A multiported memory has several access paths for read and/or write operations. This can be used effectively for a queue, where a packet can be stored at the end of the queue at the same time a packet is being read from the front of the queue. This type of operation should be used carefully since unpredictable results can occur when a multiported memory is both reading and writing the same memory location simultaneously.

For ease in both analysis and memory configuration, it is assumed that each memory chip will contain 1024 32-bit words. All memories are interleaved to achieve the bandwidth needed to allow a packet to be read or written in a single access. When a multiported memory can help the operation of the idealized systems, its effect will be considered.

<u>Control Units</u>

In the data flow systems described in this paper, a control unit refers to the logic which supervises and operates the memory units. The main functions of a control unit is to issue read or write commands to memory and to maintain status information about that memory. In an interleaved memory, the amount of logic in the control unit increases linearly with the number of memory banks, but because the logic works in parallel, the time through a control unit remains constant[2]. On multiported memories, separate control logic within the same control unit is used for each port. Typically one port is used for reading, while another port is used for writing.

Memory units being used as buffers are treated as queues. One implementation of a queue uses pointers to the first and last items in the queue; the control unit must increment or decrement these pointers as packets are stored and retrieved from the queue. A packet counter for the number of items in the queue insures that null records are not read from the queue. This can be implemented in such a way that there is no interaction between the control logic that reads from memory and the control logic that writes to memory.

---

[2] Personal communication with Dr. Louis Johnson, Dept. of Electrical Engineering, Oklahoma State University, (1982).

If all buffers use destructive readout, the control unit actions are simplified. The active packets in the buffer are the only non-zero locations in memory since memory is cleared as packets are read. When reading a packet to be forwarded to the next unit, it is checked for zeroes in all bits. If the packet does contain all zeroes, the buffer is empty and the packet is not forwarded. When a packet is stored, the pointer to the last item in the queue is incremented after the write has been initiated. When a packet is retrieved, the pointer to the next item in the queue is decremented after the read operation has been initiated. If the packet is null, the pointer to the next item in the queue must be incremented since the front of the queue did not change. In this implementation, there is no interaction between different sets of control logic; consequently, reading and writing to memory can be done concurrently.

In the idealized systems, the buffers are large enough to prevent buffer overflow, so control units will not check for this condition. The purpose of buffers is to hold incoming packets until a resource is available to accept the next packet, so buffer control units both accept packets and send packets at the same time. Packets are stored as they are sent to the queue. When a resource becomes free, a packet is read from the buffer and sent to the resource. Parallel logic within the control unit allows concurrent activity.

The timing in the control unit varies according to the amount of logic in the unit. Because the presence of an address is sufficient to initiate read or write operations, a gate delay of one is used for timing in the initiation of a read or write operation. Testing specific bit fields can be done with a single gate delay. Whole fields of words also can be tested within a single gate delay when the testing is done in parallel. Specific control unit logic for the idealized systems is given later in this chapter.

Arbitration and Distribution Networks

The arbitration and distribution units are used in data flow computers to help to route packets to the correct modules within the interconnection network and to the proper components within the modules. In the simplest case, a distribution unit has one input line and two output lines; the simplest arbitration unit will have two input lines and one output line. If the number of input lines or fan-in is fi, and the number of output lines or fan-out in a routing unit is fo, $fi > fo$ means the unit is an arbitration unit, and $fi \leq fo$ means the unit is a distribution unit. It should be understood that the bandwidth of each communication line matches the size of the particular type of packet traversing the arbitration or distribution unit. Certain fields within the packet are checked to determine the routing within the unit.

Examples of the arbitration and distribution units used in the three idealized systems are shown in Figure 11. If there are more than two input lines or two output lines, several layers of simple switches are used in the unit; this allows several packets to reside in the unit at any one time. Multiple layer switches have a timing of

$$T.ad = \log_2(fi) + \log_2(fo) \quad (3.3)$$

Single packets entering an arbitration unit pass directly through the unit, but if several packets arrive at the same time, they pass one at a time through the layers of switches, in a non-deterministic manner. Using components with two input and two output lines, an arbitration or distribution unit can be built using only

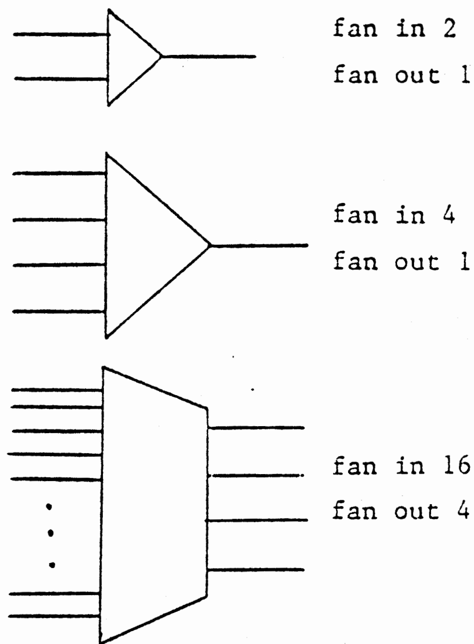
$$N.ad.layers = \left\lceil \log_2 \left\lceil \#inputs / \#outputs \right\rceil \right\rceil \quad (3.4)$$

layers of switches [30]. The use of parallel layers of switches allows maximum concurrency within the unit.

Idealized System Configurations

Each system can be described using the four components just presented. The description of each idealized system shows the arrangement of processing elements, memory units, control units, and arbitration and distribution units. The packet size and format for each system is given for both result and instruction packets. The size of destination fields and operation code fields

Arbitration



Distribution

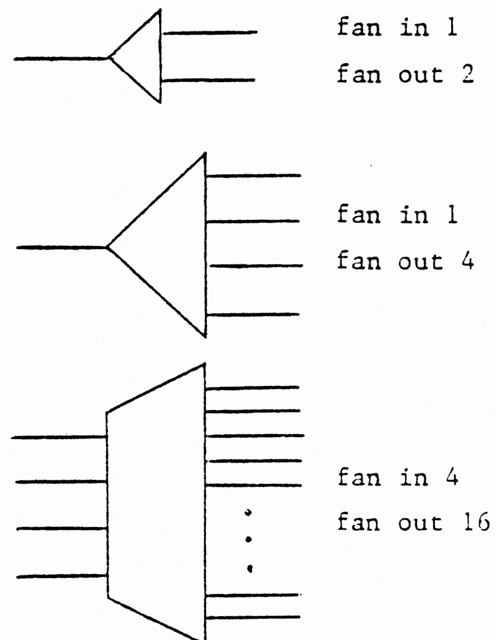


Figure 11. Arbitration and Distribution Units

in the packet comes for the original designs from each system. Packet size dictates the bandwidth of lines between different modules of the system. The interleaved arrangement of memory will be shown along with memory unit sizes; the control logic for memory units and associated components is described also.

All of the figures used in describing the idealized systems use the symbols from Figure 10. In 1971, a notation was introduced by Bell and Newell for general descriptions of computer system configurations [10]. The notation, called PMS, provides a systematic way of describing component features; four of the primitive PMS components correspond to the components used here. Appendix A contains PMS descriptions for each of the three idealized systems.

MIT Idealized System

The idealized MIT configuration is shown in Figure 12. Each processing element is connected both to the arbitration unit and to the distribution unit; each cell in the cell memory is also connected to both units. Packet formats are given in Figure 13 for both result and instruction packets.

A result packet contains a 32-bit operand and a destination address. A destination address of 18-bits can address over 250,000 instructions. An additional 14-bits are available for control information. Using full word

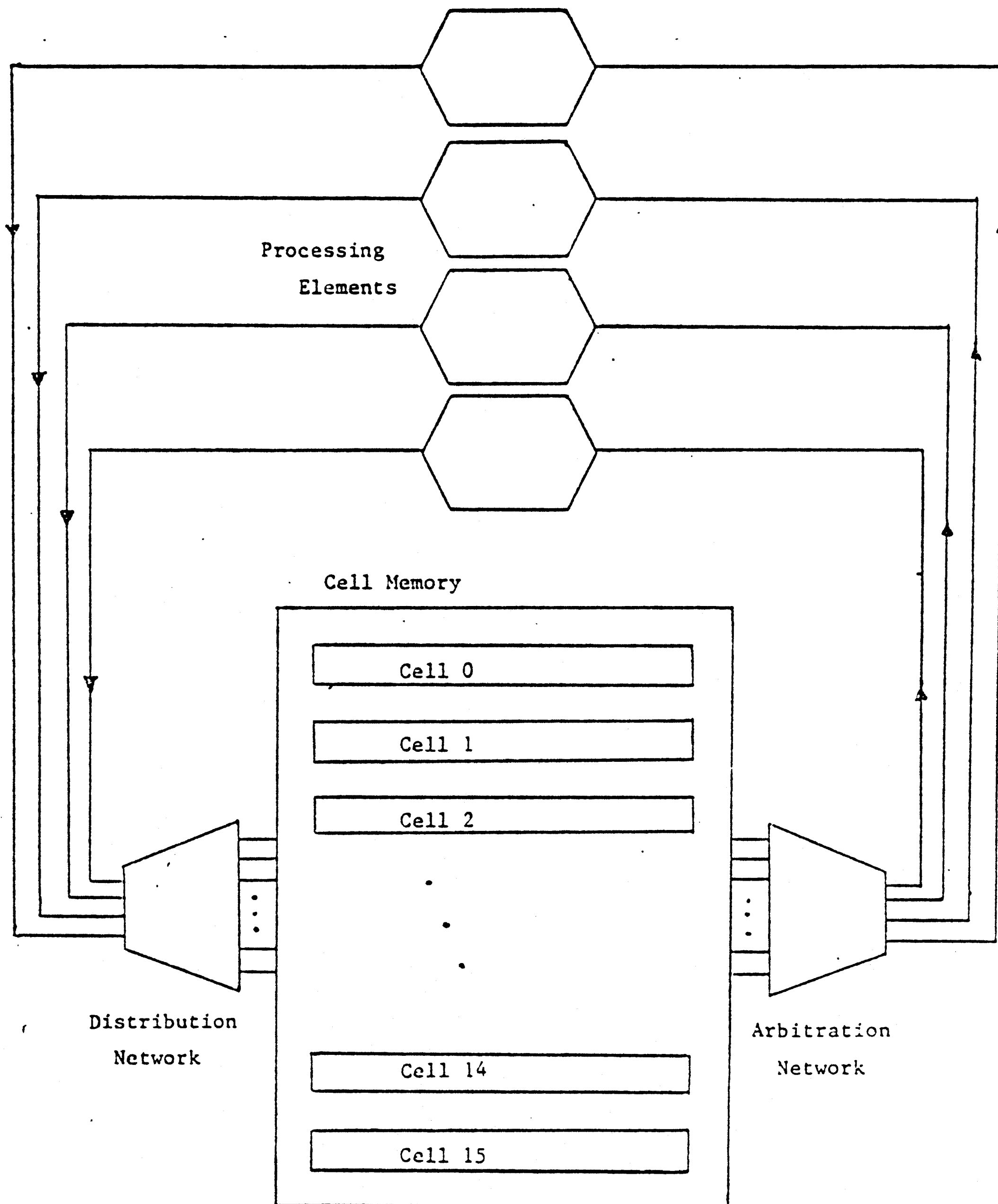


Figure 12. MIT Idealized System

(32-bit) memory access, a 64-bit packet can be accessed from memory using two-way interleaving. Figure 14 shows the structure of interleaved memory in the MIT system.

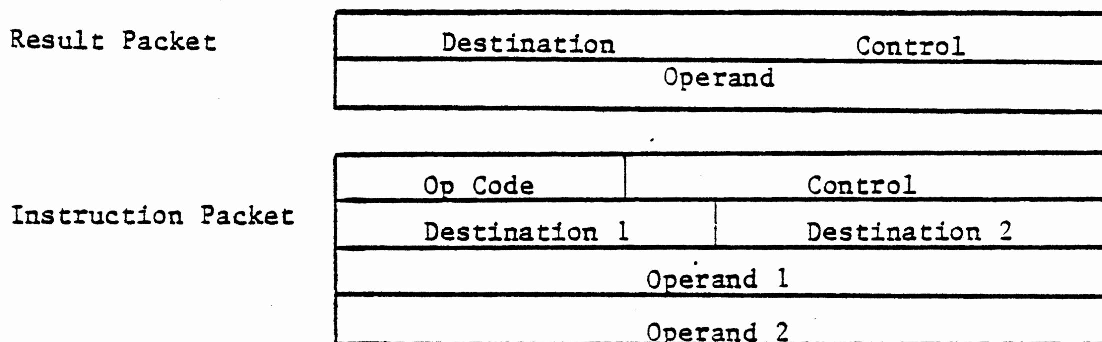


Figure 13. MIT Packet Formats

An instruction packet contains a 12-bit operation code, two 18-bit destination addresses, and two 32-bit operands. Up to 16-bits are available for other control information. The size of the operation code field, 12-bits, comes from the MIT packets described in [19]. To read a 128-bit instruction packet from memory in one memory access requires four-way interleaving.

The cell memory is divided into 16 sub-units called cells with both a read port and a write port for each cell. Each cell has the configuration shown in Figure 15 using eight 32-bit x 1K memory chips, divided into four separate banks for interleaving. This results in 2,048

instructions per cell, or 32,768 instructions in the entire cell memory.

Each cell in the cell memory contains a control unit. The control unit receives the incoming result packet and stores the operand. The control information in the instruction contains bit fields which indicate when an operand is received. After an operand is received, an appropriate bit is set to indicate that the operand has been received. If no more operands are needed, the instruction packet can be sent on to the arbitration unit. If a multiported memory is used and the control unit can read in one memory bank while writing to another bank, then the operand store can be overlapped with the checking process for the number of operands needed. This is only useful for instructions which are not enabled.

The enabling cycle time is defined as the time for an instruction to be enabled plus the time for a packet to pass through the arbitration and distribution units plus the time for a packet to pass through the PE buffer and the processing element, and the time to go through the memory buffer. Using different logic in the control unit can decrease the time required to enable an instruction. This is the logic that will be used here. After a result operand arrives at the control unit, the control unit reads the instruction first, then decrements the number of needed operands. If the number of needed operands is zero, the instruction can then be forwarded immediately;

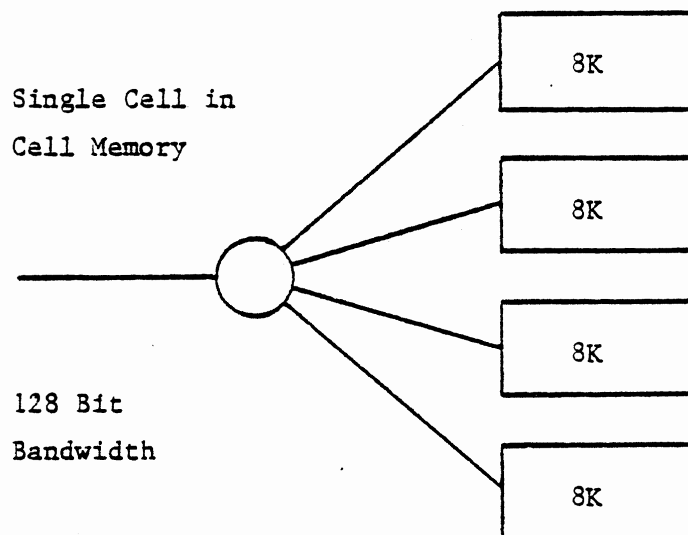


Figure 14. MIT Interleaved Memory

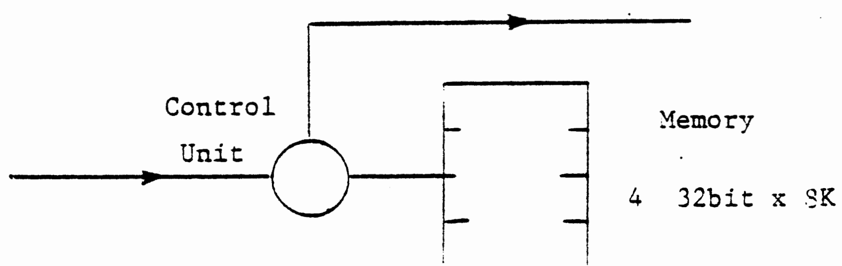


Figure 15. Individual Cell in Cell Memory

otherwise, the instruction is re-stored in the memory unit. Instructions receiving the first of two operands must wait for the second operand, so re-storing the instruction does not effect the speed of enabling instructions.

Two timings are given for packets in the cell memory: one for instructions being enabled, and one for instructions which must wait for another operand. When an instruction receives all of its operands, it is enabled. The control logic for enabled instructions includes issuing a read command, testing the number of operands needed, and sending the packet to the arbitration unit. Logic for non-enabled instructions includes a read command, testing the number of operands, and a store command. Non-enabled instructions require an extra memory access for storing the instruction back into memory. The timing equations for the cell memory is

$$T.enabled = T.mu + 4 = 44 \quad (3.5)$$

$$T.nonenabled = 2 * T.mu + 4 = 84. \quad (3.6)$$

Enabled instructions are held in the arbitration unit until a processing element becomes available.

The arbitration unit which is between the processing elements and cell memory has sixteen input lines and four

output lines. Using equation 3.3, the timing for the arbitration unit is

$$T.ad = \log(16) + \log(4)$$
$$= 6.$$

The distribution unit has four input lines and sixteen output lines. Its timing is the same as the timing for the arbitration unit. The timing for the processing elements is defined in equation 3.2, (20 gate delays).

The enabling cycle time is defined as the time to enable an instruction in the cell memory plus the time for a packet to move through the arbitration and distribution units plus the time to execute the instruction. The equation for this is

$$T.enable.cycle = T.enabled + 2*T.ad + T.pe \qquad (3.7)$$

Specific enabled cycle timing will be given later for each idealized system.

<u>Manchester Idealized System</u>

The modules of the Manchester idealized system are shown in Figure 16. The idealized system described here has four processing elements and 512K of memory.

Packet formats for the Manchester design are shown in Figure 17 for both result and instruction packets. Result packets contain an 18-bit destination address, a 32-bit operand, and a 36-bit label field, and 10 bits are

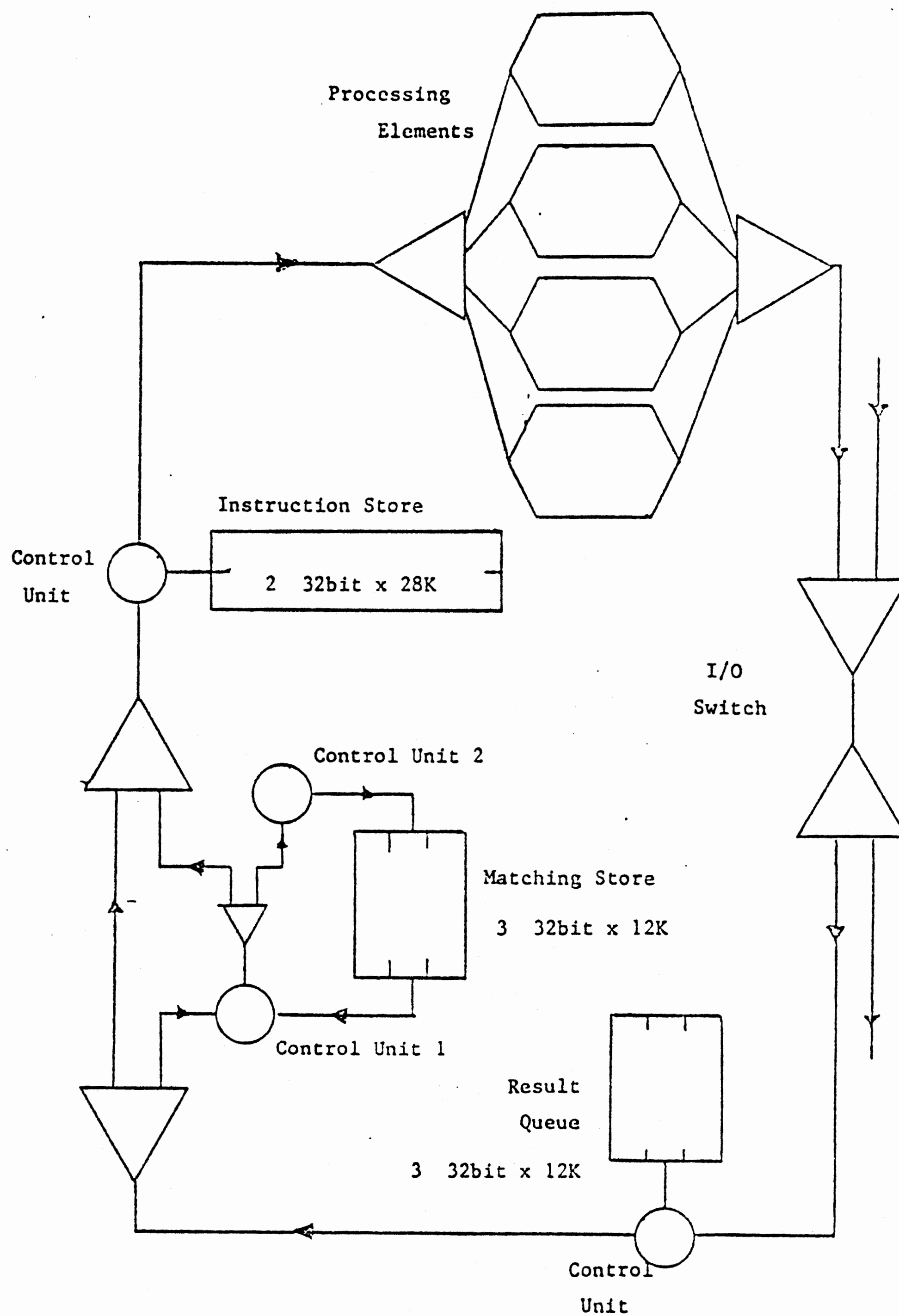


Figure 16. Manchester Idealized System

available for operand type and control information. Using full word memory access, the 96-bit result packet can be interleaved three-way in both the result queue and the matching store unit as seen in Figure 18.

Instructions held in the instruction store are partial instruction packets which contain a 12-bit operation code, two 18-bit destination addresses, and 16 control bits. Instruction packets also contain two 32-bit operands. The idealized Manchester system contains 36 32-bit x 1K memory chips in the result queue, 36 32-bit x 1K chips in the matching store, and 56 32-bit x 1K chips in the instruction store unit.

In this configuration there is space for 12,888 result packets in the result queue, 12,288 result packets in the matching store, and 28,672 (partial) instruction packets in the instruction store.

There are different types of control units in the result queue, matching store, and instruction store. The control unit in the result queue must accept incoming packets and place them at the end of the queue. When the matching store is ready to accept packets, the result queue control unit forwards the first packet available in the queue.

Using a multiported memory, the result queue control unit will operate as follows. As packets enter from the I/O switch, packets are stored at the end of the queue. As packets are stored, the pointer must be incremented to

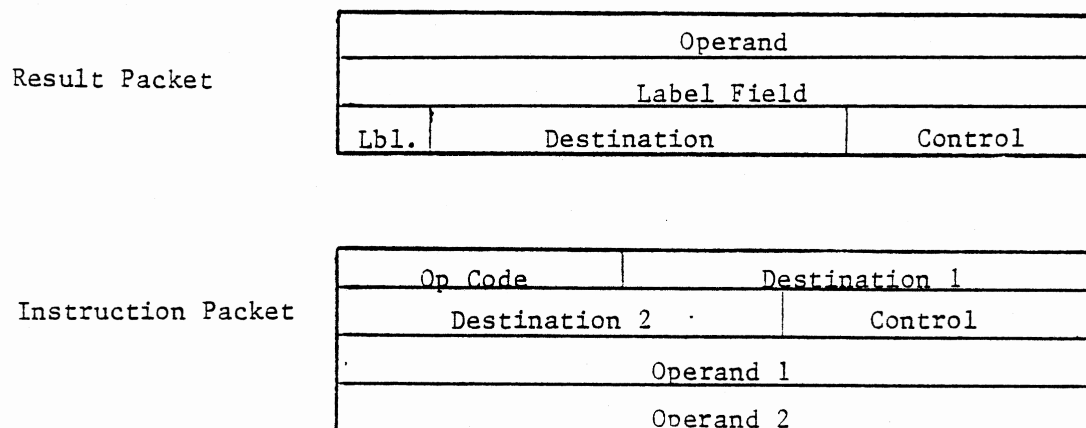


Figure 17. Manchester Packet Formats

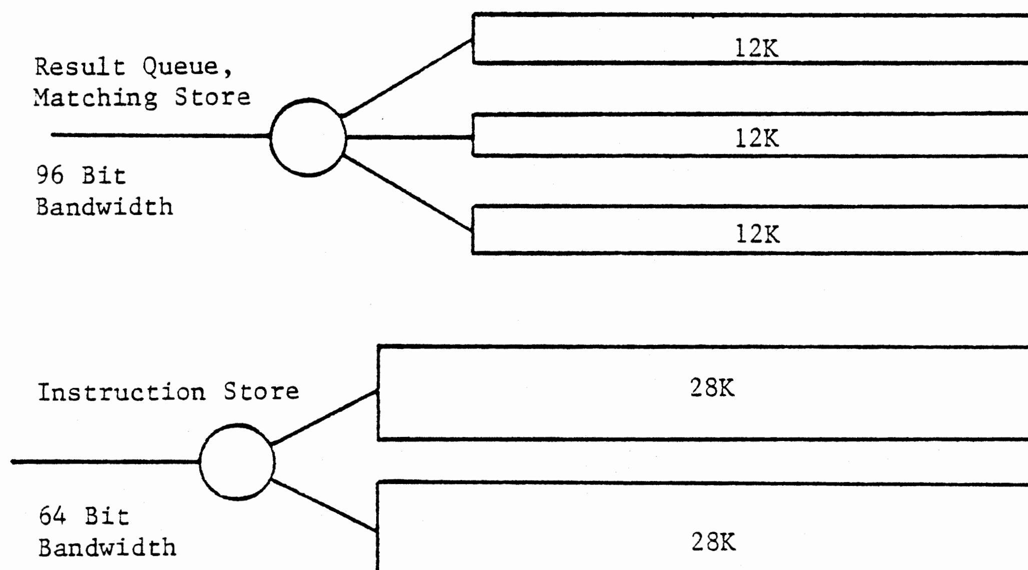


Figure 18. Manchester Interleaved Memory

the next location. The time to increment this pointer is ignored since it can be overlapped with the store operation and takes less time than the store operation. At the same time as packets are being stored, packets can be read from the front of the queue and forwarded to the matching store. A pointer must also be maintained to mark the next packet location and can be decremented while the read operation takes place.

A destructive readout memory can be used to insure that memory not being used contains zeroes. When a packet is read, it is tested to check whether it is all zeroes. If so, the packet is discarded since it is not valid. Valid packets are forwarded to the matching store.

The equation for timing in the result queue includes two memory accesses: one for storing the packet and one for reading the packet. When the packet is stored, the control logic issues a read command; when a packet is read, the control logic tests for a null packet, and forwards the packet, if possible. The control unit is given a timing of four in the result queue. The equation for the result queue timing is

$$T.rq = 2 * T.mu + 4 \qquad (3.8)$$
$$= 84.$$

The matching store contains two control units: one for reading from the memory and one for writing to the memory. The use of a multiported memory is indicated in

[28]. Packets first entering the matching store encounter a distribution unit; unary operands bypass the matching store while others are passed to the first control unit. The control unit uses the destination address and label to read from the memory. If a destructive readout memory is used, and if all zeroes are read, then no matching operand is available, so this operand must wait for the matching operand. The zero packet is discarded, and the result packet is sent to the second control unit where it is written into the memory.

There are two timing figures for the matching store: one for enabled instructions and one for nonenabled instructions. The first control unit issues a read command to the memory, then forwards the packet to the next component. If there is a zero packet in the operand pair after the read, the distribution unit forwards the packet to the second control unit. The second control unit only needs to issue a write command to the memory. The timing for the first control unit is 2 gate delays; the timing for the second control unit is only 1 gate delay. Including arbitration and distribution units, the timing equation for an enabled instruction through the matching store is

$$
\begin{aligned}
T.enabled &= T.mu + 5 \\
&= 45.
\end{aligned}
\tag{3.9}
$$

The timing for an operand that must be stored into the memory is

$$T.nonenabled = 2 * T.mu + 5 \quad (3.10)$$
$$= 85.$$

After operands are matched, they are forwarded to the instruction store unit. The operands are held in a small control unit buffer while the instruction is being read. Once the instruction is available, it is sent together with its operands to the next available processing element. The control unit in the instruction store issues a read command, then forwards the packet to the first available processing element. The timing for the instruction store control unit is 2 gate delays. The timing for the instruction store is

$$T.is = T.mu + 2 \quad (3.11)$$
$$= 42.$$

The timing for the processing elements is 20 gate delays, as in the other two systems. The timing for the arbitration and distribution units that are in front and in back of the PE's have a timing of 2 gate delays because the distribution unit has a fan-in of one and a fan-out of four, and the arbitration unit has a fan-in of four and a fan-out of one.

The time for the enabling cycle is defined as the time for an instruction to be enabled in the matching

store, plus the time to go through the instruction store, PE arbitration and distribution units, processing element, I/O switch, and the result queue. The enabling cycle timing is

$$\begin{aligned} T.enable.cycle &= T.enabled + T.is + 2*T.ad.pe \\ &\quad + T.pe + T.io + T.rq \qquad (3.12) \\ &= 197. \end{aligned}$$

Texas Instruments Idealized System

The Texas Instruments design utilizes several operational units, each unit containing both a processing element and a local memory. Several operational units are connected around a main ring. One operational unit in the idealized configuration is shown in Figure 19.

Result packets contain a 16-bit destination address and a 32-bit operand; 16 additional bits are available for control information. Using full word memory access, the 64-bit result packet can be read or written in a two-way interleaved memory.

Instruction packets contain a 16-bit operation code and a field of 4-bits for indication of the number of operands. Also, there are two 16-bit destinations and two 32-bit operands, leaving 12 bits for control information. The 128-bit packet can be accessed in a single read or write operation with a four-way interleaved memory. Result and instruction packet formats are shown in Figure 20.

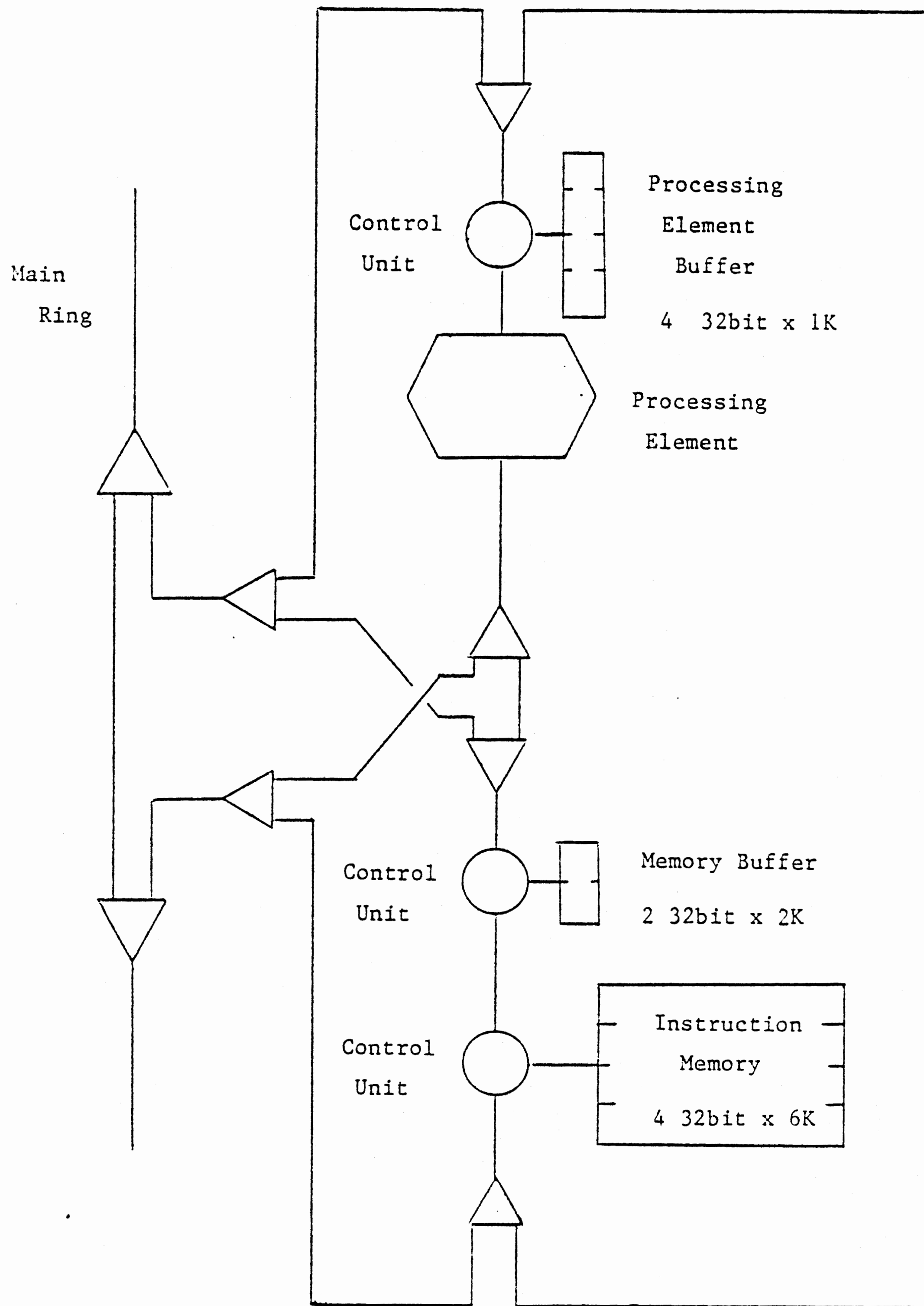


Figure 19. Texas Instruments Idealized System

Interleaved memory structures for the TI system are shown in Figure 21. In the idealized configuration there are 24 32-bit x 1K memory chips in the memory unit and four 32-bit x 1K memory chips in both buffers. Because the PE buffer is interleaved four-way and the memory buffer is only interleaved two-way, the memory buffer can hold twice as many packets as the PE buffer. This is consistent with the fact that one instruction can generate two result packets.

Using the packet and memory size, it can be seen that in each operational unit there is space for 6,144 possible instruction packets in the instruction memory, 1,024 result packets in the PE buffer, and 2,048 result packets in the memory buffer. For four operational units, this gives a maximum of 28,672 instruction packets, and 8,192 result packets.

The are several possible strategies of operation in the instruction memory. In the interest of faster instruction enabling, the instruction packet for the operand is first read into the control unit when a result operand is received. The number of operands needed is decremented and tested. If this number is zero, then the instruction packet is enabled and forwarded from the instruction memory. If the number of needed operands is not zero, then the instruction packet is stored back into the instruction memory. Depending on the result destinations in the packet, the packet is either sent out of the operational unit or sent back to the PE buffer.

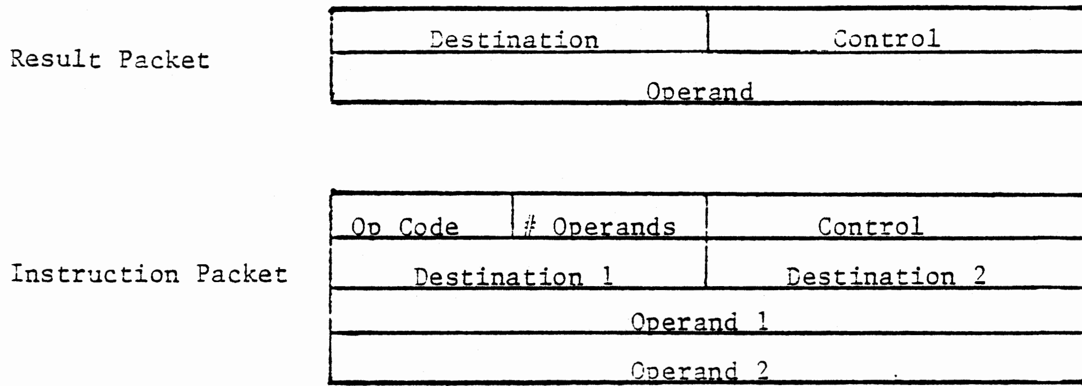


Figure 20. TI Packet Formats

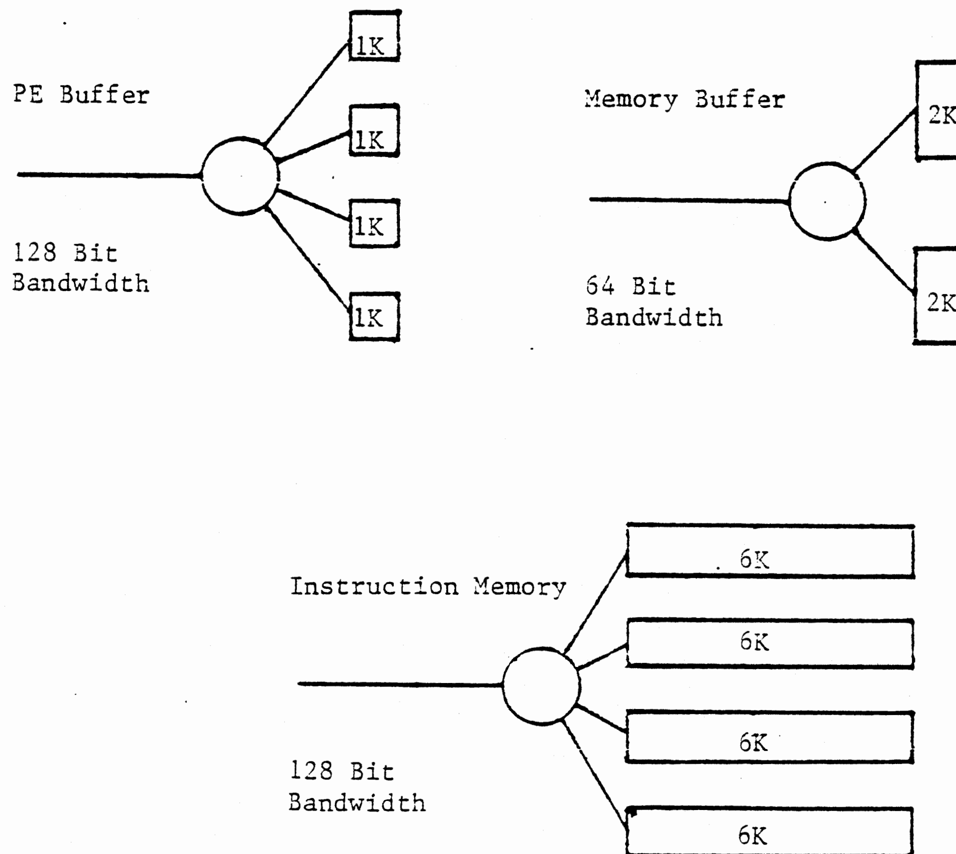


Figure 21. TI Interleaved Memory

In the Texas Instruments idealized system, there are two different types of control units. One control unit is used with the memory unit and controls the matching of instruction operands. Both buffer units can use the same type of control unit for control of packets that are held within the buffer. The actions of buffers used in the TI DDP are not described in the literature available from the company, so a queue is used in this analysis. The buffer control units operate similarly to the control unit associated with the Manchester result queue. As described earlier, a buffer of this nature require a pointer to the first and last items in the queue; reading and writing packets requires the control unit to maintain these pointers.

Both buffers in the system operate the same way. When packets arrive in the buffer control unit, they are placed at the back of the queue; the control unit can update the pointer to the next store location when the packet is being stored. As stated before, the time required to update the pointer can be ignored since a memory access in the idealized system is slower than the time required to update the pointer. While packets are being stored at the end of the queue, packets can be sent out from the front of the queue. Once a packet is read from memory, it is checked to see whether it is zero. A zero packet occurs whenever the buffer is empty and is discarded. Valid packets are sent to the next unit in the interconnection network.

The only difference between the PE buffer and memory buffer is the interleaving control; the PE buffer unit is four-way interleaved, and the memory buffer is two-way interleaved.

The time for a packet to pass through the two buffer control units include a read and write operation as packets are stored and read from the buffer. The control logic issues a store command as the packets are received. When packets are read out, the logic issues a read command, tests for a null packet, and sends the packet to the next module. Control unit timing is four gate delays, and the buffer timing is

$$T.bu = 2 * T.mu + 4 = 84. \tag{3.13}$$

The instruction memory can be analyzed both with a timing for enabling instructions, and with a timing for the storing of operands. The logic needed to enable an instruction includes issuing a read command, logic to decrement the number of operands yet required, and logic to forward the packet from the instruction memory if the operand count is zero. The enabled instruction timing is

$$T.enabled = T.mu + 4 = 44. \tag{3.14}$$

The timing for storing operands back into memory includes the time required for an extra memory access, and

the time required for the control logic to issue a store command in lieu of forwarding the packet. The timing for a non-enabling instruction is

$$T.nonenabled = 2 * T.mu + 4 \tag{3.15}$$
$$= 84.$$

Timing for the processing elements is the same as the other processing elements: 20 gate delays as stated in equation 3.2.

Because of the arrangement of modules on the TI system, the enabling cycle includes only one processor. The logic to effect the enabling cycle is contained entirely within the same operational unit; its timing formula is:

$$T.enable.cycle = T.enabled + 4*T.ad + 2*T.bu + T.pe \tag{3.16}$$

Enabled cycle timing for the idealized systems is given in the next chapter. The timing for every module in each of the three idealized systems are in Appendix C.

CHAPTER IV

ANALYSIS OF IDEALIZED DATA FLOW SYSTEMS

In analyzing the different idealized data flow systems, two measures are used: the time required to move an enabled instruction completely around the interconnection network, and the rate in units of packets per million gate delays at which packets move through the processing elements. If there are no bottlenecks in the system, the maximum system throughput is the rate of operation of the processing elements when they are operating at their maximum rate. Any modules operating at a rate less than the processing elements is a bottleneck in the system.

Equations to describe the rates for each module are given in this chapter. Specific values for the idealized system being analyzed are given as well. The form used for rate equations is:

R.x = Rate equation for module x (Eq#)
= Value for idealized system.

All timing and rate equations are listed together in Appendix D.

Packet Rates Through Interconnection Network Modules

As stated before, it is desirable to achieve a linear increase in computing power as processors are added into the system. The packet rate is based on the ratio of the number of parallel components (modules) to the component (module) timing. The packet rate for component (module) x is defined as

$$R.x = (N.x / T.x)$$

in units of packets per million gate delays (PMG). In rate equations, N.x is the number of modules, components, or operations that can execute in parallel, and T.x is the timing for x as calculated in Chapter III.

In deriving the packet rates for arbitration and distribution units, it was found that regardless of the fan-in or fan-out of the unit, the packet rate is one packet per gate delay when the unit operates at its maximum rate. This result is a consequence of building the arbitration and distribution units in parallel stages from simple one gate switches and a consequence of packets residing in each stage in the unit during packet transfer. Because the number of stages always equals the number of gate delays, all arbitration and distribution units will have a rate of

$$R.ad = (N.ad.stages / T.ad) \quad (4.1)$$
$$= 1{,}000{,}000 \text{ packets per million gate delays}$$
$$= 1{,}000{,}000 \text{ PMG.}$$

Using a processing element time of 20 gate delays, the idealized systems will have the following rate for four processors

$$R.pe = (N.pe / T.pe) \quad (4.2)$$
$$= 200{,}000 \text{ PMG.}$$

The processing element rate is not necessarily the same as the system rate, because some other module may be a bottleneck.

An enabling rate is given for each system using the enabling cycle timing from Chapter III. Programs that are strictly serial execute only one instruction per enabling cycle. Because of this, the enabling rate is defined as

$$R.enable = (1 / T.enabled.cycle) \quad (4.3)$$

The enabling rate is the minimum rate at which instructions can execute.

MIT Idealized System Rates

The rates for the processing elements, the arbitration unit, and the distribution unit have been given previously. The rate of the arbitration and

distribution units was defined as 1,000,000 PMG in equation 4.1, this applies to both the MIT arbitration and distribution units. In equation 4.2, the rate of four processing elements is given as 200,000 PMG.

In modules where enabling takes place, the rate of the module is derived from a non-weighted average of the enabling instruction timing and the nonenabling instruction timing. The ratio of nonenabling operands to enabling operands is program dependent; an average is used in comparing all systems. If all instructions are binary instructions, the average used in the enabling timing is exact. An average of the enabling and nonenabling times from equations 3.5 and 3.6 is used in the cell memory rate; these times are 44 gate delays and 84 gate delays, respectively. The cell memory in the MIT design has sixteen cells in the cell memory that can operate in parallel, giving a cell memory rate of

$$R.cm = (N.cells / ((T.enabled + T.nonenabled)/2) = 250,000 \text{ PMG.} \quad (4.4)$$

Using the enabling cycle timing of 76 gate delays from equation 3.7, the MIT enabling rate is

$$R.enable = (1 / T.enabled.cycle) = 13,157 \text{ PMG.} \quad (4.5)$$

Manchester Idealized System Rates

The rates for the processing elements are stated in equation 4.2. The rate of the four processing elements on the Manchester system is 200,000 PMG. Since the I/O switch is composed solely of arbitration and distribution units, it uses the arbitration and distribution rates from equation 4.1 or 1,000,000 PMG.

The Manchester result queue was designed to allow reading and writing to be done concurrently. The two parallel units are used in the rate calculation. Using the result queue timing of 84 gate delays from equation 3.8, the result queue rate is

$$R.rq = (N.rq / T.rq) \quad (4.6)$$
$$= 23,809 \text{ PMG.}$$

The ratio of unary instructions to binary instructions is program dependent; to be comparable with other memories where enabling takes place, the average of enabling and nonenabling instructions is used for the timing in the matching store. The control units in the matching store operate in parallel. By using the enabling and nonenabling timing from equations 3.9 and 3.10 (45 gate delays and 85 gate delays), the matching store rate is computed to be

$$R.ms = (N.ms / ((T.enabl+T.nonenabl)/2)) \quad (4.7)$$
$$= 30,769 \text{ PMG.}$$

The instruction store operates as a serial memory using one control unit attached to the memory. Using the timing of 42 gate delays from equation 3.11, the instruction store rate is

$$R.is = (N.is / T.is) \qquad (4.8)$$
$$= 23{,}809 \text{ PMG.}$$

The arbitration and distribution units around the PE have a rate of 1,000,000 PMG, (equation 4.1).

Using the enabling cycle timing of 197 gate delays from equation 3.12, the Manchester enabling rate is

$$R.enable = (1 / T.enabled.cycle) \qquad (4.9)$$
$$= 5{,}076 \text{ PMG.}$$

Texas Instruments Idealized System Rates

The rate used for the processing elements in the Texas Instruments design is not the same as the other PE rates, because a single PE is used in each operational unit. All other rates for the TI system are for a single operational unit also. Therefore, the PE rate for the Texas Instruments idealized system is calculated as 50,000 PMG from equation 4.2. The overall TI processing element rate is 200,000 for four processors. The timing for all arbitration and distribution units will have a rate of 1,000,000 PMG as computed from equation 4.1.

The buffers in the TI system were designed so that reading and writing could be done in parallel. Consequently, two units are used in the rate calculation. Using the buffer timing of 84 gate delays computed from equation 3.13, the buffer rate is

$$R.bu = (N.bu / T.bu) = 23{,}809 \text{ PMG.} \quad (4.10)$$

Each TI instruction memory is a serial memory. The memory units act in parallel without interference from one another. Because the instruction memory is accepting enabled and non-enabled instructions, a non-weighted average of enabled and non-enabled instruction timing is used for the timing in the instruction memory. Using the enabling and nonenabling timing of 44 gate delays and 84 gate delays from equations 3.14 and 3.15, the instruction memory has a rate of

$$R.im = (N.im / ((T.enabl+T.nonenabl)/2) = 16{,}000 \text{ PMG.} \quad (4.11)$$

All arbitration and distribution units each have a rate of 1,000,000 PMG as defined in equation 4.1.

Using the enabling cycle timing of 236 gate delays from equation 3.16, the Texas Instruments enabling rate is

$$R.enable = (1 / T.enabled.cycle) = 4{,}237 \text{ PMG.} \quad (4.12)$$

All rates are shown in Tables III-V.

TABLE III

MIT IDEALIZED MODULE TIMING AND RATE FIGURES

Module		Timing (Gate delays)	Packet Rate (PMG)
Cell Memory	T.cm		250,000
enabled	T.enabled	44	
nonenabled	T.nonenabled	84	
Arbitration Unit	T.ad	6	1,000,000
Distribution Unit	T.ad	6	1,000,000
Processing Elements (4)	T.pe	20	200,000
Enabling Cycle Rate	T.enable.cycle		13,157

TABLE IV

MANCHESTER IDEALIZED MODULE TIMING AND RATE FIGURES

Module		Timing (Gate delays)	Packet Rate (PMG)
I/O Switch	T.ios	2	1,000,000
Result Queue	T.rq	84	23,809
Matching Store	T.ms		30,769
enabled	T.enabled	45	
nonenabled	T.nonenabled	85	
Instruction Store	T.is	42	23,809
PE Arbitration and Distribution Unit	T.ad	2	1,000,000
Processing Elements (4)	T.pe	20	200,000
Enabling Cycle Rate	T.enable.cycle		5,076

TABLE V

TEXAS INSTRUMENTS IDEALIZED MODULE
TIMING AND RATE FIGURES

Module		Timing (Gate delays)	Packet Rate (PMG)
Arbitration and Distribution Units	T.ad	1	1,000,000
Buffer Units	T.bu	84	23,809
Instruction Memory	T.im		15,625
enabled	T.enabled	44	
nonenabled	T.nonenabled	84	
Processing Element (1)	T.pe	20	50,000
Enabling Cycle Rate	T.enabled.cycle		4,237

Analysis Of Packet Rates

Maximizing throughput of the processing elements is desired in the analysis of packet rates. Any modules of the system with a rate lower than the processing element rate are considered a potential bottleneck in the system. If bottlenecks are present in any of the idealized systems, possible improvements are explored in an attempt to remove the bottleneck.

In the idealized MIT system, the arbitration and distribution units operate at a rate higher than the PE's. This is true in the other two designs as well. More importantly, the cell memory rate exceeds the processing element rate which results in the maximum possible

throughput rate through the PE's. If the number of processors were to be raised, the number of cells in the memory also would have to increase in order to match the needed PE rate. The design of the MIT system allows for both a variable number of PE's and a variable number of cells in the cell memory.

If the packet rate from the processors were to exceed one million packets per million gates, the arbitration and distribution units would then become a bottleneck. A data flow system of twenty or more processing elements could have this problem. Larger arbitration and distribution units would be needed if a larger system were to be built. The use of several parallel arbitration or distribution units instead of a single arbitration and distribution could eliminate this bottleneck as well.

It appears that the MIT idealized system could operate at the maximum throughput rate through the processing elements; larger systems could also be built to fully utilize the processing element throughput rate.

In the idealized Manchester system, several modules appear to be bottlenecks. The rates of the result queue, the matching store, and the instruction store, are all less than the rate required by the processors. If these rates were used in a data flow system, the processing elements would be idle a large percentage of the time.

In order to determine the increase in rate needed of the modules which are bottlenecks, a ratio of the

processing element rate to the rate of the module which is a bottleneck can be used. The speed-up rate can be calculated by dividing the rate of the processing elements, by the rate of the module which is a bottleneck.

$$R.x.su = (R.pe / R.x) \tag{4.13}$$

The instruction store would need to operate 9 times faster to keep the processing elements fully utilized. The matching store would have to operate 7 times faster to match the PE rate. And the result queue would need to operate 9 times faster to match the PE rate.

One possible solution to the bottlenecks would be the use of a higher speed memory; the rates of all units using a memory would then be increased. This is only a temporary solution and depends on the number of processors in the system. A data flow system with a large number of processors would require an extremely fast memory if all modules are to match the PE rate; this might not be practical for a large system. There is no way for a memory operating serially to match the rate of a set of parallel processors, especially if there are a large number of processing elements. In a specific configuration where the number of processors is constant, it might be possible to find a memory which is of a specific technology that can support the processing elements. A serial memory preceeding a set of parallel processors restricts the number of processors it can service effectively.

The Texas Instruments design is unique in two ways. First, while the processors do operate in parallel, each processing element has separate sets of memory logic and control logic. The problem of a memory servicing a set of parallel processors is not present here because each processing element has a local memory. A serial memory can service a single processing element adequately; this is evident in von Neumann computers. In the idealized system, the buffer units and instruction memory rates are not great enough to keep the processing elements busy at all times. A memory operating 3 times faster than the idealized memory would increase the rate of the buffer units and instruction memory enough to match the PE rate. The number of processors do not effect the required memory speed, so finding a particular memory technology to match the PE rate is sufficient regardless of the size of the system.

Both partitioning of the data flow graph into subgraphs and the number of operational units on the main ring affect packet throughput. The literature available on the TI DDP suggests that the main ring traffic would be light; because of this, the analysis here ignores activity on the main ring. As the number of subgraphs of the program increases, the number of packets passed among subgraph partitions should increase. The time a packet spends traversing the main ring is directly proportional to the number of processors on the ring. If the number of

processing elements is large, then the problems of data flow graph partitioning and operational unit communication will degrade total system performance. A large number of processors on the ring also increases ring contention and slows down main ring traffic even further. For a small number of processors, the TI design appears to be competitive with the other data flow architectures.

Another measure used in the idealized systems is the enabling cycle time to complete a circuit in the interconnection network. In highly serial programs where only one instruction can only execute at a time, the time for program execution would be closer to the enabling rate than the maximum rate through the processing elements. In the MIT design, the enabling rate is greater because there are no buffers or queues which lengthen the enabling cycle. The Manchester design contains one queue which degrade the performance of t he enabling cycle. The Texas Instruments design contains two buffers which degrades the enabling cycle. Serial programs execute in a time proportional to the enabling cycle. If a program contains little parallelism, the advantage of a high parallel processing rate cannot be utilized.

Advantages and Disadvantages of Selected Data Flow Architectures

The organization and speed of memory causes most of the bottlenecks in the idealized systems. If a serial memory operates in front of a set of parallel processors, then only a limited number of processors will be to utilized fully. The use of a memory operating in parallel appears to be well suited to a data flow system with parallel processors.

The length of the enabling cycle can be a detriment for serial programs. The enabling cycle greatly reduced the possible maximum throughput rate of the data flow system for serial programs in both the Manchester and Texas Instruments designs.

The use of serial packet routing also can cause a reduction in packet rate, as it does in the TI main ring. If the rate of packet transfer on the main ring is slower than the rate of modules in the operational units, then the processing elements are idle a larger percentage of time when they are waiting for packets from other operational units.

In the attempt to arrive at comparable data flow systems, some features of the MIT and Manchester architectures were deleted. One of the most recent of the MIT data flow systems includes a secondary memory system;

no other data flow designs have suggested using anything but a main memory. The secondary memory is not designed to be used for file storage, but rather it is designed to be used for instructions that are not currently being referenced. It also may be used as a part of a virtual memory system in a multiprogrammed system. The cell memory is similiar to a cache in such a system. Possible motivations include improved memory speed or the desire to build a smaller cell memory on large systems.

In the extended Manchester design, many layers of operational units are connected together. A single layer, or operational unit, is similiar to the system described here; an exchange switch replaces the I/O switch for layer-to-layer communication. The Manchester design might appear to be similiar to the TI design with several operational units, but it has an improved communication routing system. The exchange switch can be built using arbitration and distribution units in stages and would process packets in logarithmic time.

Real Machine Comparison

Using the rates derived above, the timing in terms of gate delays can be compared to the timing of a real machine in terms of million instructions per second. The timing from the Cray 1 computer was chosen for the basis of comparison, because the Cray 1 has one of the highest instruction throughput rates. The Cray 1 was designed

using small scale integrated circuitry; all chips are bipolar with gates having a fan-in/fan-out of four or five. Register chips operate with a 6 nano-second cycle time, and memory chips operate with a 50 nano-second cycle time. Because the idealized memory is twice as slow as the processing element timing, assume the Cray 1 processor timing (add time) is 25 nano-seconds for the purposes of this analysis. Using the ratio of gate delays per nano-second, the timing used in the idealized system can be converted to million instructions per second (MIPS).

The timing used in the processing elements is 20 gate delays, which corresponds to the 25 nano-second time on the Cray 1 computer; therefore 1 gate delay equals 1.25 nano-seconds. The rate of the processing elements is 200,000 instructions per million gate delays, or 1 instruction per 6.25 nano-seconds. This gives an approximate rate of 160 MIPS using four processors. In contrast, the Cray 1 can operate between 125 - 250 million floating point operations per second (MFLOPS), and is able to sustain a rate of 138 MFLOPS [44]. Also, the Cray 1 does operations on 64-bit words, while the idealized systems used 32-bit words. Because of the difference in units, MIPS to MFLOPS, the difference in word sizes, and the assumtion of a 25 nano-second CRAY 1 add time, this cannot be a precise comparison. But the resulting idealized rate of 160 MIPS does indicate that the computation rate is within range of a present day

supercomputer. A data flow system with more processing elements, or with faster technology, could achieve a processing rate far above any systems built today.

CHAPTER V

SUMMARY, CONCLUSIONS, AND SUGGESTED FUTURE RESEARCH

Summary and Conclusions

A timing analysis of the architecture of three data flow computers and their interconnection networks were presented. The main thrust of analysis was directed toward the systems from MIT, Texas Instruments, and the University of Manchester. The different modules which are present in the three data flow systems were examined, and the timings for single packets to pass through the different modules were derived. Packet structure, packet contents, and memory configuration were described for all three systems. The three data flow systems were then configured in ideal systems so they could be compared.

No particular technology was used in analyzing the different systems. Four basic components were used to construct the different modules of the system. The components used include: processing elements, memory units, arbitration and distribution units, and control

units. The timing of each component was expressed in terms of the gate delays it requires for a packet to pass through the component; this was done to aid in a technology independent analysis. Once the timings for the modules were derived, the maximum packet rates through the modules followed logically. Then packet rates were used in analyzing the performance of the interconnection networks.

In analyzing the idealized systems, two measures were used: the maximum packet rate and the minimum packet rate through the different systems. The goal for the processing element rate was to obtain the maximum packet rate possible. The enabling cycle was used as the minimum rate; this occurs when highly serial programs are executed.

With parallel processors, the processing rate increases linearly with the number of processors executing in parallel. In the various idealized system rates, the module with the lowest rate represents the maximum processing rate because the module with the lowest rate is the worst bottleneck of the system. Any modules with a rate less than that of the processing elements was considered to be a bottleneck in the system. Both arbitration and distribution units operated at a rate higher than the processing elements and thus did not effect processor utilization. Memory units that operate serially were definite bottlenecks, especially when

servicing parallel processing elements. Memory units operating in parallel can achieve rates that are greater than or equal to the processing element rate.

The minimum processing rate is the enabling rate, which occurs during execution of sequential programs. The enabling rate was found to be lower in systems with multiple memory units. This was caused by the reading and writing of packets into the different memory units. The more memory modules in which a single packet is stored, the longer the enabling cycle, thus reducing the enabling rate for serial programs. When executing sequential programs on a data flow system, a high percentage of the available processors may not be used at any given instant. In all three idealized systems, the enabling rate was less than 10 percent of the maximum processing element rate. Because the enabling rate does not depend on the number of processors, the enabling rate is constant for a given system.

The rates derived in Chapter IV demonstrate that the architectures from the University of Manchester and Texas Instruments cannot operate at the maximum processing rate. Both designs use a memory operating serially, and this appears to be the major bottleneck. The MIT design uses a memory with parallel access, and thus the memory can be configured so the memory rate matches the processing rate.

Overall, it appears that the MIT architecture is the best of the three data flow designs. The processing

element rate was matched by all other modules in the system and had no specifically iden tifiable bottlenecks. The MIT system also has an architecture that could be expanded into a larger system with no module bottlenecks.

Suggested Future Research

In evaluating the packet rates for the idealized systems, program behavior was ignored. Future research analyzing data flow processor rates could analyze program characteristics to determine enabling rates more carefully. Simulation could also be used to determine the processor utilization for different types of programs.

Additional research comparing data flow systems can analyze the extended architecture designs. This would include the use of a secondary memory as either an instruction store or a virtual memory. The effects of communication between operational units could be examined.

This thesis examined three specific data flow systems, future research could compare other data flow systems as well. All three systems examined were data driven systems, future work might compare data driven systems with demand driven systems.

The increased use of complex computer systems and large complex applications undoubtedly will continue. Advances in computer architecture, hardware, and software, will be required to meet the rising demand for computing power.

SELECTED BIBLIOGRAPHY

[1] Ackerman, William B., "Data Flow Languages." Proceedings of the 8th Annual Symposium on Computer Architecture (May, 1981), 1-28.

[2] Ackerman, William B., and Dennis, Jack B., "VAL - A Value Oriented Algorithmic Language, Preliminary Reference Manual." M.I.T. Laboratory for Computer Science TR-218, Cambridge, Mass. (June, 1979).

[3] Agerwala, Tilak, and Arvind, K., "Data Flow Systems." Computer (February, 1982), 10-13.

[4] Arvind, Kathail V., "A Multiple Processor Data Flow Machine That Supports Generalized Procedures." Proceedings of the 8th Annual Symposium on Computer Architecture (May 12, 1981), 291-302.

[5] Arvind, K., and Gostelow, Kim P., "A Computer Capable of Exchanging Processing Elements for Time." Technical Report 77, University of California, Irvine, Ca. (January, 1976), 1-57.

[6] Arvind, K., Gostelow, Kim P., and Plouffe, Wil, "An Asynchronous Programming Language and Computing Machine." Department of Computer Science, University of California, Irvine, Ca. (December 8, 1978), 1.1-2.16.

[7] Arvind, K., and Pingali, K., "The U-Interpreter." Department of Computer Science, University of California, Irvine, Ca. (February 26, 1981), 1-12.

[8] Arvind, K., and Pignali, K., "A Data Flow Architecture with Tagged Tokens." Tutorial Notes, 8th Annual Symposium on Computer Architecture, Minneapolis, Minn. (May 12, 1981).

[9] Barnes, G. H., Brown, R. M., Kato, M., Kuck, D. J., Slotnick, D. L., and Stokes, R. A., "The ILLIAC IV Computer." IEEE Transactions on Computers, Vol. C-17, No. 8 (August, 1968), 746-757.

[10] Bell, C. Gordon, and Newell, Allen, Computer Structures: Readings and Examples. Mc Graw Hill, New York, N.Y. (1971), 15-22, 615-628.

[11] Bryant, Randal E., "Simulation of Packet Communication Architecture Computer Systems." M.I.T. Laboratory for Computer Science TR-188, Canbridge, Mass. (November, 1977), 9-17.

[12] Burkowski, F. I., "A Multi-User Data Flow Architecture." Proceedings of the 8th Annual Symposium on Computer Architecture (May 13, 1981), 327-340.

[13] Cornish, Merrill, The TI Data Flow Architecture: The Power of Concurrency for Avionics. Texas Instruments Inc., Austin, Texas (1979).

[14] Davis, A. L., "The Architecture and System Model of DDM1: A Recursively Structured Data Driven Machine." 5th Annual Symposium on Computer Architecture (1978), 210-215.

[15] Davis, A. L., "A Data Flow Evaluation System Based on Concepts of Recursive Locality." AFIPS Conference Proceedings (1979), 1079-1086.

[16] Dennis, Jack B., "Segmentation and the Design of Multiprogrammed Computer Systems." Journal of the ACM, Vol. 12, No. 4 (October, 1965), 589-602.

[17] Dennis, Jack B., "First Version of a Data Flow Procedure Language." CSG Memo 93, M.I.T. Project MAC, Cambdidge, Mass. (August, 1974), 1-21.

[18] Dennis, Jack B., "Packet Communication Architecture." CSG Memo 130, M.I.T. Project MAC, Cambridge, Mass. (August, 1975), 1-6.

[19] Dennis, Jack B., "Proposed Research on Architectural Principles for Large Memory Systems." CSG Memo 132, M.I.T. Project MAC, Cambridge, Mass. (October, 1975), 1-17.

[20] Dennis, Jack B., "The Varieties of Data Flow Computers." Procedings of the 1st International Conference on Distributed Computing (October, 1975), 430-439.

[21] Dennis, Jack B., "Data Flow Supercomputers." Computer (November 1980), 48-56.

[22] Dennis, Jack B., Andrew, G., and Leung, C. K., "Building Blocks for a Data Flow Computer Prototype." Laboratory for Computer Science, M.I.T., Cambridge, Mass. (1980), 1-8.

[23] Dennis, Jack B., and Misunas, David P., "A Preliminary Architecture For a Basic Data Flow Processor." CSG Memo 102, M.I.T. Project MAC (August, 1974), 1-27.

[24] Dennis, Jack B., and Misunas, David P., "A Computer Architecture for Highly Parallel Signal Processing." Proceedings of the 1974 ACM National Conference, New York, NY. (November, 1974), 402-409.

[25] Dennis, Jack B., Misunas, David P., and Leung, C. K., "A Highly Parallel Processor Using a Data Flow Machine Language." CSG Memo 134, M.I.T. Project MAC, Cambridge, Mass. (January, 1977), 16-49.

[26] Ellis, David J., "Formal Specifications for Packet Communication Systems." M.I.T Laboratory for Computer Science TR-184, Cambridge, Mass. (November, 1977), 9-17.

[27] Gostelow, Kim P., and Thomas, Robert E., "Performance of a Simulated Data Flow Computer." IEEE Transactions on Computers, Vol. C-29, No. 10 (October, 1980), 905-919.

[28] Gurd, John, Watson, Ian, and Glauert, John, "A Multilayered Data Flow Computer Architecture." Department of Computer Science, University of Manchester, Manchester, England (March, 1980), 1-56.

[29] Hintz, R. G., and Tate, D. P., "Control Data STAR-100 Processor Design." Compcon 72, IEEE Computer Society Conference Proceedings (September, 1972), 1-4.

[30] Jacobson, Robert G., and Misunas, David P., "Analysis of Structures for Packet Communication." CSG Memo 151, M.I.T. Project MAC, Cambridge, Mass. (August, 1975), 1-6.

[31] Jensen, John C., "Basic Program Representation in the Texas Instruments Data Flow Test Bed Compiler." Preliminary Report, Texas Instruments Inc., Austin, Tex. (January 29, 1980), 1-20.

[32] Karp, R. M., and Miller, R. E., "Properties of a Model for Parallel Computation: Determinacy, Termination, and Queuing." SIAM Journal of Applied Mathematics, Vol. 14, No. 6 (November, 1966), 1390-1411.

[33] Keller, Robert M., Lindstrom, Gary, and Patil, Suhas, "An Architecture for a Loosely-Coupled Parallel Processor." University of Utah, UUCS-78-105 (October, 1978).

[34] Kuck, David J., The Structure of Computers and Computation. Wiley and Sons, New York, NY. (1978), 483-488.

[35] Leung, Clement Kin Cho, "Fault Tolerance in Packet Communication Computer Architecture." M.I.T. Laboratory for Computer Science TR-250, Cambridge, Mass. (September, 1980), 131-154.

[36] Maekawa, M., "Optimal Processor Interconnection Topologies." Proceedings of the 8th Annual Symposium on Computer Architecture (May 14, 1981), 171-186.

[37] Misunas, David P., "Deadlock Avoidance in a Data Flow Architecture." CSG Memo 116, M.I.T. Project MAC, Cambridge, Mass. (February, 1975), 1-7.

[38] Oldehoeft, Aurthur E., Thoreson, S., Retnadhas C., and Zingg, R., "The Design of a Software Simulator for a Data Flow Computer." Department of Computer Science, Iowa State University, Ames, Ia. (1977), 1-23.

[39] Oldehoeft, Aurthur E., Zingg, R., and Retnadhas, C., "Measurement of Parallelism in Computer Programs Through Analysis of Program Graphs." Department of Computer Science, Iowa State University, Ames, Ia. (1979), 1-15.

[40] Organick, Elliot I., "New Directions in Comuter Systems Architecture." Department of Computer Science, University of Utah (1978), 1-4.

[41] Osborne, Adam, Jacobson, S., and Kane, J., An Introduction to Microcomputers, Volume II: Some Real Products. Osborne and Associates Inc., Berkeley Ca. (1977).

[42] Plas, A., et. al., "LAU-System Architecture: A Parallel Data Driven Processor Based on Single Assignment." Proceedings of the International Conference on Parallel Processing (1976), 293-302.

[43] Rumbaugh, James E., "A Data Flow Multiprocessor." IEEE Transactions on Computers, Vol. C-26, No. 2 (February, 1977), 138-146.

[44] Russell, Richard M., "The CRAY-1 Computer System." Communications of the ACM, Vol. 21, No. 1 (January, 1978), 63-72.

[45] Sauber, William B. ed., A Data Flow Architecture Implementation. Texas Instruments Inc., Austin, Tex. (1980), 1-20.

[46] Srini, V. P., "An Architecture for Extended Abstract Data Flow." Proceedings of the 8th Annual Symposium on Computer Architecture (May 13, 1981), 303-326.

[47] Traub, A. H., Collected Works of John von Neumann: Volume 5. MacMillan Co., New York, N.Y. (1963), 1-79.

[48] Treleaven, P. C., Brownbridge, D., and Hopkins, R. "Data Driven and Demand Driven Data Flow Computers." Computing Surveys, Vol. 14, No. 1 (March, 1982), 94-144.

[49] Watson, Ian, and Gurd, John, "A Prototype Data Flow Computer with Token Labeling." Proceedings of the 1979 National Computer Conference (1979), 623-628.

[50] Watson, Ian, and Gurd, John, "A Practical Data Flow Computer." Computer (February, 1982), 51-57.

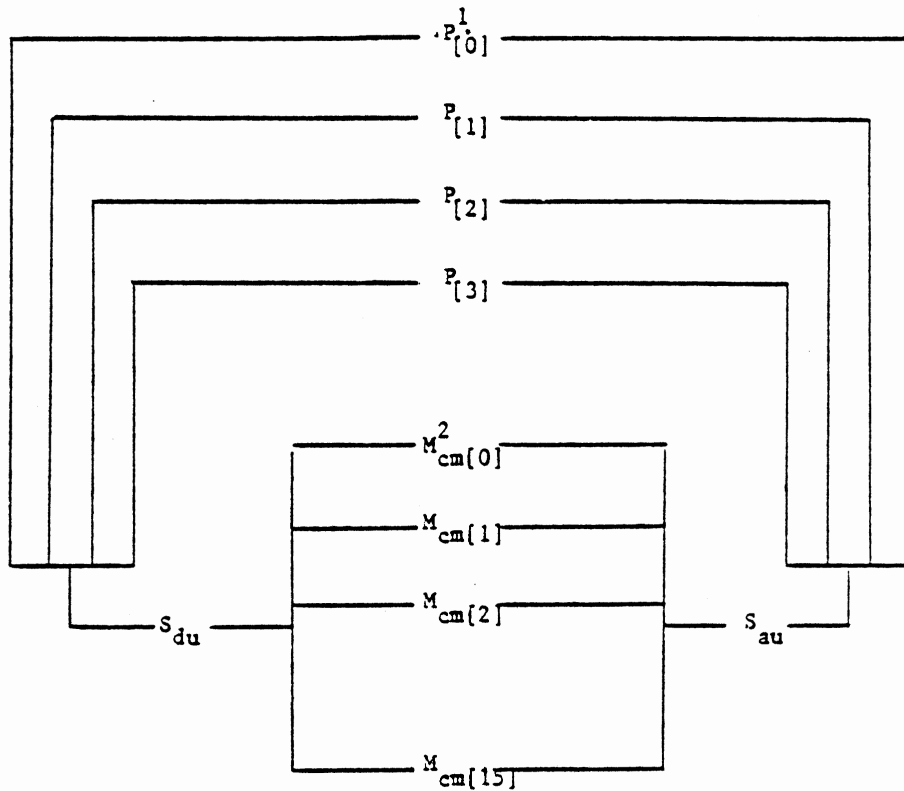
APPENDIX A

PMS DESCRIPTIONS OF THE THREE IDEALIZED DATA FLOW SYSTEMS

PMS Descriptions

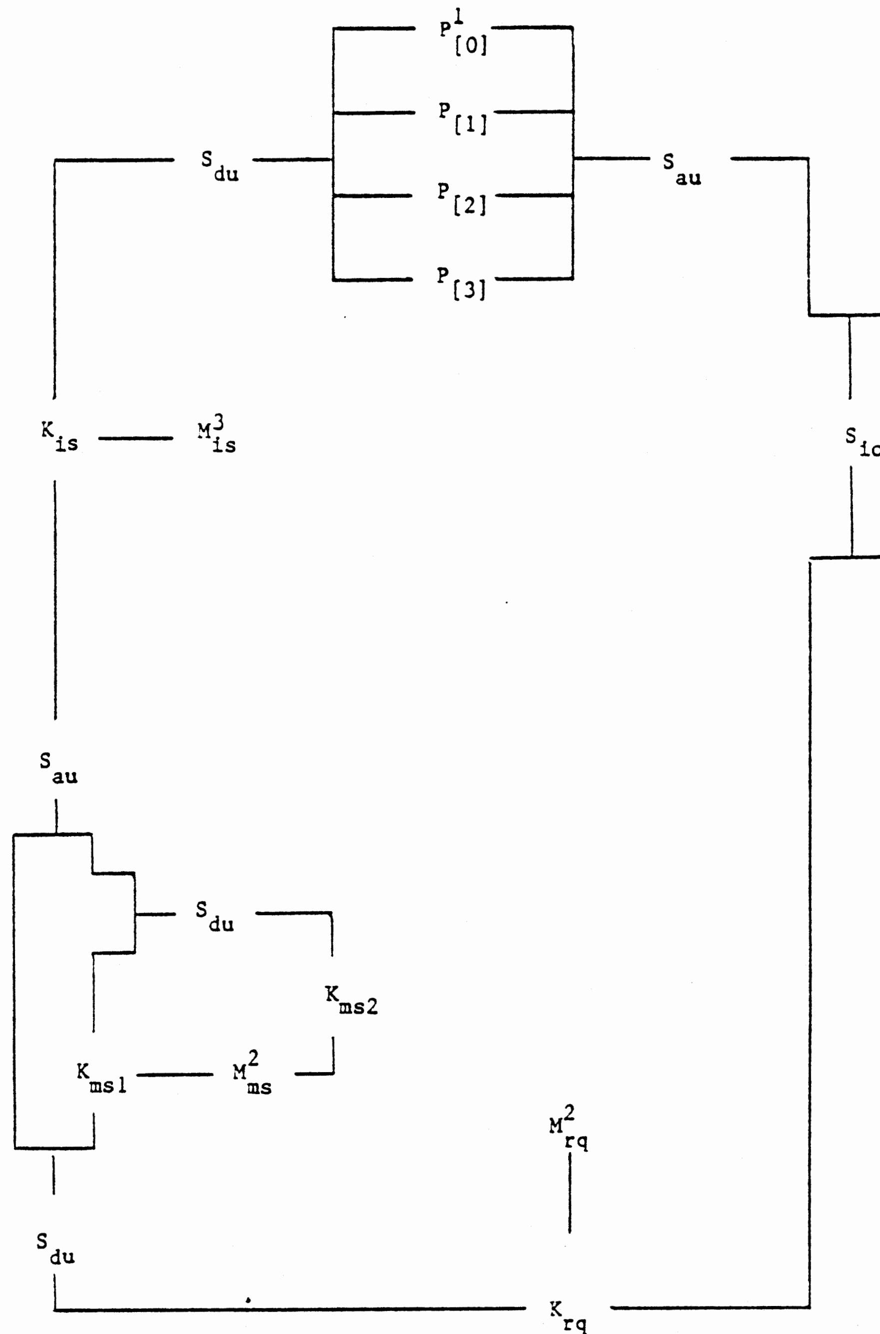
PMS is a notation used to describe the overall structure of computer systems. Components and modules of a system are represented by several basic symbols, and the different control and data paths are shown as the links between the components and modules. The five symbols used here are: P - processor, M - memory, S - switch, C - computer, K - control unit. Subscripted identifiers are used to identify specific components and modules; abbreviations that are used conform to the terms used in the idealized systems. Subscripts within square brackets indicate a set of identical components or modules. Superscripts are used for matching the symbols with the legend.

For some components and modules, there is a more detailed description in the legend giving several attributes and specific values for those attributes. For example: X(a1:v1,a2:v2, . . . ,aN:vN) gives the values v1,v2, . . . ,vN, to the attributes a1,a2, . . . ,aN for component or module X. For a complete description of PMS notation see [10].



1. P (cycle time:20 gate delays,data path:32 bits)
2. M_{cm} (size:32K 32 bit words,cycle time:40 gate delays, data transfer:128 bits)

Figure 22. MIT Idealized Data Flow System

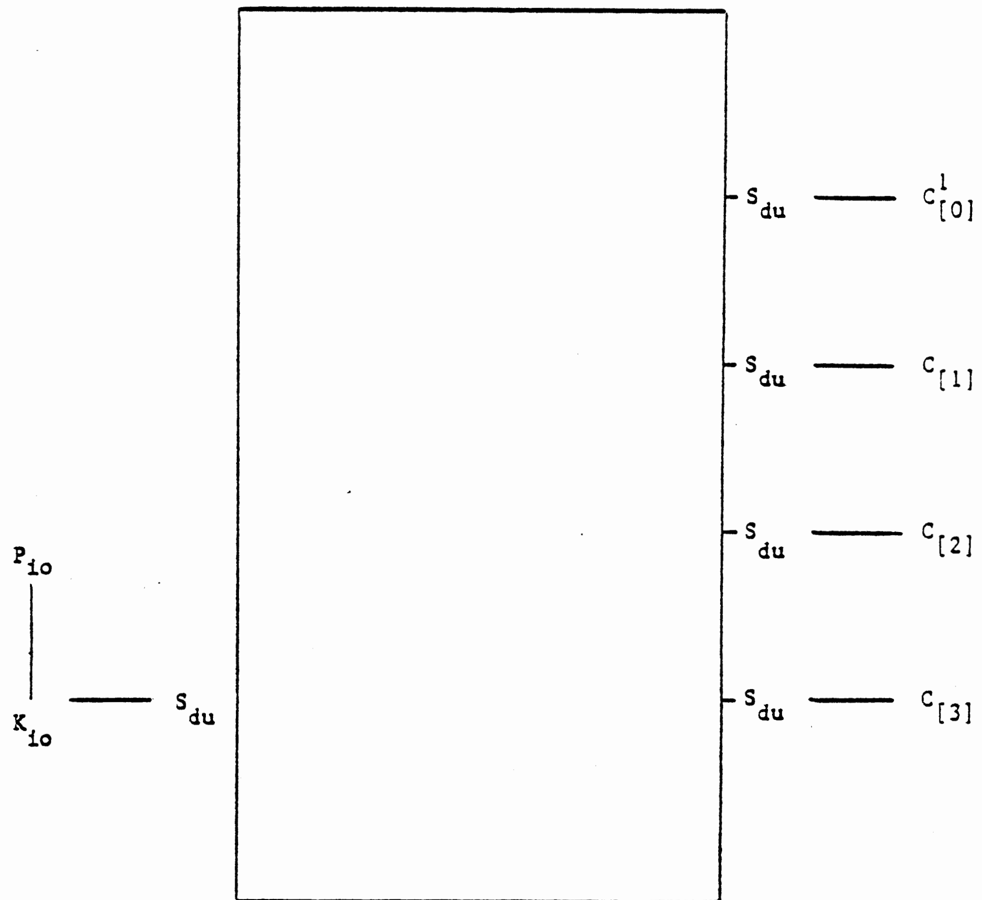


1. P (cycle time:20 gate delays,data path:32 bits)

2. $M_{rq,ms}$ (size:36K 32 bit words, cycle time:40 gate delays, data transfer: 96 bits)

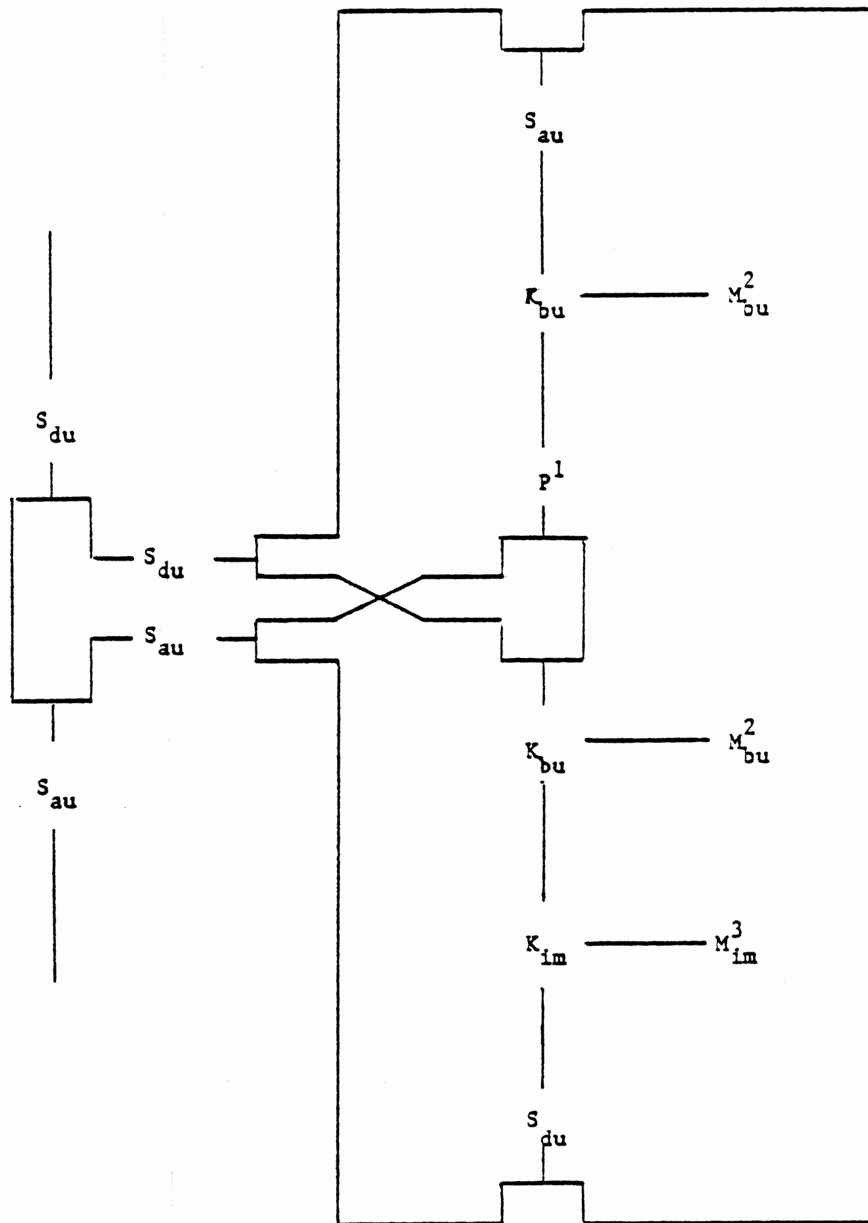
3. M_{is} (size:56K 32 bit words,cycle time:40 gate delays, data transfer:64 bits)

Figure 23. Manchester Idealized Data Flow System



1. C - Operational unit: see Figure 25

Figure 24. Texas Instruments Idealized Data Flow System



1. P (cycle time:20 gate delays,data path:32 bits)
2. M_{bu} (size:4K 32 bit words,cycle time:40 gate delays, data transfer:128 bits)
3. M_{im} (size:24K 32 bit words,cycle time:40 gate delays, data transfer:128 bits)

Figure 25. Texas Instruments Single Operational Unit

APPENDIX B

TIMING FOR SINGLE PACKETS TO PASS THROUGH IDEALIZED INTERCONNECTION MODULES

TABLE VI

MIT IDEALIZED SYSTEM COMPONENT AND MODULE TIMING

Module/Component	Actions	Timing
Cell Memory		
Control unit	initiate read operation	1
	decrement # of operands	1
	test if # operands = 0	1
	forward instruction packet	1
-or-	initiate store operation	
Memory	read/store operation	40
		----
	enabled instr.	44
	nonenabled instr.	84
Arbitration and Distribution Units	route packets	6
Processing Element	process instruction	20

TABLE VII

MANCHESTER IDEALIZED SYSTEM COMPONENT AND MODULE TIMING

Module/Component		Actions	Timing
I/O Switch			
Arbitration and distribution unit			
		route packets	2
Request Queue			
Control unit		initiate read operation	1
		increment pointer	*
		initiate read operation	1
		decrement pointer	*
		test if null packet	1
		forward packet	1
	-or-	increment pointer	
Memory unit		read/store operation	80

			84
Matching Store			
Distribution unit		route packet (2)	2
Control unit 1		initiate read operation	1
		forward packet	1
Memory unit		read/store operation	40
Arbitration unit		route packet	1
Control unit 2		initiate store operation **	1
			----
		enabled instr.	45
		nonenabled instr. **	86
Instruction Store			
Control unit		initiate read operation	1
		forward packet	1
Memory unit		read operation	40

			42
PE Arbitration and Distribution Unit		route packets	2
Processing element			20

* Timing is zero since operation is overlapped with memory access.

TABLE VIII

TEXAS INSTRUMENTS IDEALIZED SYSTEM
COMPONENT AND MODULE TIMING

Module/Component		Actions	Timing
Ring Switch			
Arbitration and distribution units			
		route packets	2
Buffer Unit			
Control unit		initiate write operation	1
		increment pointer	*
		initiate read operation	1
		decrement pointer	*
		test if null packet	1
		forward packet	1
	-or-	incrememt pointer	
Memory unit		read/store operation	80

			84
Instruction Memory			
Control unit		initiate read operation	1
		decrement # of operands needed	1
		test if it equals zero	1
		if yes forward packet	1
	-or-	store operand and # operands	
Memory unit		read/store operation	40
			----
		enabled instr.	44
		nonenabled instr.	84
Processing Element		process instruction	20

* Timing is zero since operation is overlapped with memory access.

APPENDIX C

LIST OF SYMBOLS

Symbols Used in Idealized Data Flow System Analysis

Symbol		Description
fi	-	Fan in
fo	-	Fan out
ad	-	Arbitration and distribution networks
cu	-	Control unit
mu	-	Memory unit
pe	-	Processing element
cm	-	Cell memory (MIT)
rq	-	Result Queue (Manchester)
ms	-	Matching store (Manchester)
is	-	Instruction store (Manchester)
enabled	-	Enabled instruction
nonenabled	-	Non enabled instruction
T.add	-	Time to perform addition
T.pe	-	Average time through processing element
T.ad	-	Time through arbitration/distribution network
T.mu	-	Time through memory unit
T.cu	-	Time through control unit
T.cm	-	Time through MIT cell memory
T.rq	-	Time through Manchester request queue
T.ms	-	Time through Manchester matching store
T.is	-	Time through Manchester instruction store

T.pe.ad	-	Time through Manchester processing element arbitration and distribution networks
T.bu	-	Time through TI buffer unit
T.enabled	-	Time for instruction to be enabled
T.nonenabled	-	Time for nonenabled instuction to be stored
N.pe	-	Number of processing elements
N.ad	-	Number of parallel stages in arbitration and distribution units
N.cm	-	Number of cells in cell memory (MIT)
N.rq	-	Number of parallel control units on result queue (Manchester)
N.ms	-	Number of parallel control units on matching store (Manchester)
N.is	-	Number of parallel control units on instruction store (Manchester)
N.bu	-	Number of parallel control units on buffer unit (Texas Instruments)
N.mu	-	Number of parallel memories in memory unit (Texas Instruments)
R.pe	-	Packet rate through processing elements
R.ad	-	Packet rate through arbitration and distribution networks
R.mu	-	Packet rate through memory unit
R.cm	-	Packet rate through M.I.T. cell memory
R.rq	-	Packet rate through Manchester result queue
R.ms	-	Packet rate through Manchester matching store
R.is	-	Packet rate through Manchester instruction store
R.enable	-	Packet rate for enabled instructions

APPENDIX D

LIST OF EQUATIONS

Equations Used in Packet Rate Analysis

Equation #	Equation
Timing Equations	
3.1	$T.add \leq 1/2 * (5 + \log_f t) * \log_2 t + 4$
3.2	$T.mu = 2 * T.pe$
3.3	$T.ad = \log_2(fi) + \log_2(fo)$
3.4	$N.ad.layers = \lceil \log_2 \lceil \#inputs / \#outputs \rceil \rceil$
MIT Idealized System Timing Equations	
3.5	$T.enabled = T.mu + 4$
3.6	$T.nonenabled = 2 * T.mu + 4$
3.7	$T.enable.cycle = T.enabled + 2*T.ad + T.pe$
Manchester Idealized System Timing Equations	
3.8	$T.rq = 2 * T.mu + 4$
3.9	$T.enabled = T.mu + 5$
3.10	$T.nonenabled = 2 * T.mu + 5$

Equation #	Equation
3.11	T.is = T.mu + 2
3.12	T.enable.cycle = T.enabled + T.is + 2*T.ad.pe + T.pe + 2*T.ad + T.rq

Texas Instruments Idealized System Timing Equations

Equation #	Equation
3.13	T.bu = 2 * T.mu + 4
3.14	T.enabled = T.mu + 4
3.15	T.nonenabled = 2 * T.mu + 4
3.16	T.enable.cycle = T.enabled + 4*T.ad + 2*T.bu + T.pe

Rate Equations

Equation #	Equation
4.1	R.ad = (N.ad.stages / T.ad)
4.2	R.pe = (N.pe / T.pe)
4.3	R.enable = (1 / T.enable.cycle)

Equation #	Equation
MIT Idealized System Rate Equations	
4.4	R.cm = (N.cells / ((T.enabled+T.nonenabled)/2))
4.5	R.enable = (1 / T.enable.cycle)
Manchester Idealized System Rate Equations	
4.6	R.rq = (N.rq / T.rq)
4.7	R.ms = (N.ms / ((T.enabled+T.nonenabled)/2))
4.8	R.is = (N.is / T.is)
4.9	R.enable = (1 / T.enable.cycle)
Texas Instruments Idealized System Rate Equations	
4.10	R.bu = (N.bu / T.bu)
4.11	R.im = (N.im / ((T.enabled+T.nonenabled)/2))
4.12	R.enable = (1 / T.enable.cycle)

APPENDIX E

TERM GLOSSARY

Term Glossary

Term **Description**

Arbitration unit

A routing unit with more input lines than output lines.

ARPANET

A loosely-coupled network in which computers across the country communicate over long distance communication lines.

Bandwidth

The number of bits of information that can be transferred over communication lines, typically the bandwidth is the same as the size of the packets traversing the lines.

Data flow computer

A data flow computer system has many processors in a tightly-coupled configuration. The machine representation of programs run on data flow systems allow instructions to operate in parallel on different processors, exploiting program parallelism.

Data driven

In data driven computers, the availability of all operands for an instruction triggers execution of that instruction.

Data path width

The number of bits of data in an operand.

Demand driven

In demand driven computers, the requirement for a result triggers the instruction that will generate that result.

Distribution unit

A routing unit with the same or more output lines than input lines.

Enabling cycle time

The enabling cycle time is defined as the time for an instruction to be enabled, plus the time to pass through the different arbitration and distribution units, plus the time through the processing element, plus the time for an operand to return to the memory unit in which enabling takes place.

ETHERNET

A computer network where many external devices within a close proximity to the main processor are connected in a local network.

Gate delay

The time for a unit of information to pass through an simple electronic device or gate.

Instruction firing (releasing, enabling)

When all operands and control information are present in an instruction, the instruction is enabled and may be released (or fired) for execution.

Interconnection network

The communication lines between the different modules and components of a computer system.

Loosly-coupled network

A multiple CPU computer system where each CPU can operate independantly of the other CPU's is loosly-coupled.

Multiple ported memory

A memory with multiple access paths for reading and/or writing to memory.

Multiple ring architecture

An architecture with several single rings for instruction execution.

Packet

A packet is a collection of information in a specified format.

Parallel program

A computer program where instructions can execute at the same time.

Routing unit

A unit with one or more input lines and one or more output lines that accepts packets and routes them to their destination.

Serial program

A computer program where only one instruction can only execute at a time.

Single ported memory

A memory with a single access path for reading and writing data.

Single ring architecture

An architecture with one circular path for instruction and data.

Tightly-coupled network

A multiple CPU computer system where the CPU's are dependant on each other for operation is tightly-coupled.

VITA

John Robert Kerns

Candidate for the Degree of

Master of Science

Thesis: PACKET COMMUNICATION WITHIN DATA FLOW COMPUTERS

Major Field: Computing and Information Sciences

Biographical:

Personal Data: Born in Des Moines, Iowa, March 3, 1956, the son of Lois M. Kerns.

Education: Graduated from Des Moines Technical High School, Des Moines, Iowa, on June 7, 1974; received Bachelor of Arts degree in Mathematics from Southwest State University, Marshall, Minnesota, on May 27, 1978; completed requirements for the Master of Science degree at Oklahoma State University, Stillwater, Oklahoma, in July, 1982.

Professional Experience: Programmer/analyst at Argonne National Laboratory; Argonne, Illinois, Summer 1977, and June, 1978 to July, 1979. Graduate teaching assistant, Department of Computing and Information Sciences; Oklahoma State University, Stillwater, Oklahoma, August, 1979 to July, 1981. Consultant at Met-Con Construction Co.; Faribault, Minnesota, January, 1980 to May, 1981. Graphics application programmer at the Center for Applications of Remote Sensing; Oklahoma State University, August, 1981 to July, 1982. Member of ACM since 1978.