71847

QUARTERLY TECHNICAL PROGRESS REPORT

July, August, September 1974

MASTER

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

DISTRIBUTION OF THIS DOCUMENT UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

QUARTERLY TECHNICAL PROGRESS REPORT

July, August, September 1974

UIUCDCS-QPR-74-3

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Energy Research and Development Administration, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

MASTER

DISTRIBUTION OF THIS DOCUMENT UNLIMITED

leg

TABLE OF CONTENTS

		Page
1.	HARDWARE RESEARCH (NAVY).....	1
1.1	STORM (Project No. 49).....	3
	1.1.1 Summary.....	3
	1.1.2 Status of Project.....	3
1.2	MOLECULAR STOCHASTICS (Project No. 51).....	3
	1.2.1 Project Summary.....	3
	1.2.2 Project Status.....	3
	1.2.3 Future Work.....	5
1.3	STOCHFEED (Project No. 53).....	5
1.4	BURSTCALC (Project No. 54).....	7
	1.4.1 Introduction.....	7
	1.4.2 Digital Multiplier.....	7
	1.4.3 Conclusions.....	11
1.5	ARC (Project No. 56).....	11
	1.5.1 Introduction.....	11
	1.5.2 General Description.....	12
	1.5.3 Damage Tolerance.....	14
	1.5.4 Project Status.....	14
	1.5.5 Future Work.....	19
1.6	FM-DIRAD (Project No. 59).....	19
	1.6.1 Introduction.....	19
	1.6.2 Scope of Work.....	19
	1.6.3 Initial Ideas.....	20
1.7	AM-DIRAD (Project No. 60).....	22
	1.7.1 Introduction.....	22
	1.7.2 Preliminary Considerations.....	22
1.8	BURSTSORT (Project No. 62).....	23
	1.8.1 Introduction.....	23
	1.8.2 Project Status.....	24
	1.8.3 Future Work.....	26
1.9*	BURSTCOMM (Project No. 63).....	26
	1.9.1 Summary.....	26
	1.9.2 Progress.....	26
2.	HARDWARE RESEARCH (VARIOUS).....	28
2.1	FROG (Project No. 36).....	29
	2.1.1 Introduction.....	29
	2.1.2 Project Status.....	29
	2.1.3 Future Work.....	29
2.2	CAECOTRON (Project No. 43).....	30
2.3	THESPIAC (Project No. 47).....	30
2.4	NORMAN (Project No. 50).....	31
2.5	NASACOMP (Project No. 57).....	32
	2.5.1 Overview.....	32
	2.5.2 Arithmetic Processing Element Design....	32
	2.5.3 Prealignment.....	36
	2.5.4 Accumulator.....	36
	2.5.5 Normalization.....	37

	Page
2.5.6 Prototypes.....	37
2.6 CARDALERT--An Arrhythmia Monitor (Project No. 64)...	39
2.6.1 Introduction.....	39
2.6.2 Specifications of CARDALERT.....	39
2.6.3 CARDALERT Block Diagram.....	39
2.6.4 The Preprocessor.....	44
3. SOFTWARE SYSTEMS RESEARCH.....	46
3.1 Numerical Processes.....	46
3.1.1 Tests of Integration Packages.....	46
3.1.2 Periodic Solutions of Ordinary Differential Equations.....	46
3.1.3 Analysis of Methods for Ordinary Differential Equations.....	47
3.1.4 Absolute Stability of Multistep Methods.....	48
3.1.5 DIFSUB and Partial Differential Equations.....	51
3.1.6 Global Error for Ordinary Differential Equations.....	52
3.2 Graphic Support Packages.....	55
3.2.1 Call OS Plot Programs.....	55
3.3 User Oriented Software.....	78
3.3.1 Global Macro Analysis.....	78
3.3.2 The PL-F Compiler.....	88
3.3.3 Network Analysis Package Modifications.....	97
3.3.4 Interpreter Language Assembler.....	101
3.4 Satellite Processor Software and Hardware.....	110
3.4.1 CTKTST - Backpanel Wiring Test Program.....	110
3.4.2 PDP-8 Software.....	118
3.4.3 PDP8 Party-line AC Expander.....	121
3.4.4 Computek Wiring Diagnostic Device.....	127
3.4.5 Accumulator Panel.....	130
4. SWITCHING THEORY AND LOGICAL DESIGN.....	131
5. MACHINE AND SOFTWARE ORGANIZATION STUDIES.....	133
5.1 Algorithm Bounds.....	133
5.2 Program Analysis.....	134
5.2.1 Speedup of Iterative Programs.....	134
5.2.2 IF Analysis.....	134
5.2.3 Machine Simulator.....	135
5.2.4 Alignment Network Study.....	135
5.3 File Processing.....	135

	Page
5.3.1 File Processing Software.....	135
5.3.2 Inverted File Retrieval System.....	136
5.3.3 An Approach to the Table Manipulator....	136
5.3.4 Floppy Disk.....	137
5.3.5 Hardware Support.....	137
5.3.6 High Speed Secondary Memory.....	138
6. ASPECTS OF COMBINATORIAL COMPUTING.....	140
7. THEORY OF DIGITAL COMPUTER ARITHMETIC.....	146
7.1 A Study of Arithmetic Structure for LSI.....	146
7.2 Continued Fraction Arithmetic.....	146
7.3 Function Evaluation Techniques.....	147
7.4 Calculator Design.....	148
8. ACSES, THE AUTOMATED COMPUTER SCIENCE EDUCATION SYSTEM ON PLATO.....	150
9. MACHINE PERCEPTION AND INTELLIGENCE.....	160
10. ITERATIVE SOLUTION OF SYMMETRIC AND NONSYMMETRIC SYSTEMS WITH DYNAMIC ESTIMATION OF PARAMETERS.....	163
11. MESH RESEARCH PROJECT.....	164
11.1 System Software.....	164
11.2 Micro-programming Languages.....	164
11.3 Experimental Network Hardware.....	165
11.3.1 Maintenance.....	165
12. GENERAL INFORMATION.....	166
12.1 Personnel.....	166
12.2 Bibliography.....	167
12.3 Abstracts.....	169
12.4 Colloquia.....	177
12.5 Drafting.....	178
12.6 Shop's Production.....	178

I. HARDWARE RESEARCH (NAVY)

(Supported in part by Navy Contract #N000 14-67-A-0305-0024.)

Summary

With the exception of Jim Cutler's continuing report on the Molecular Stochastics work, the projects written about in this section represent new work that has just started up under Navy support and is, for the most part, being carried out by new research assistants. The common theme of all these projects is research into practical uses of stochastic processing methods and, implicitly, comparison with conventional techniques. Here is a short description of the projects that are and will subsequently be reported on in this section.

Molecular Stochastics--Jim Cutler

The design of stochastic transducers, devices which measure physical parameters (temperature, flow rate, light intensity, etc.) directly in terms of stochastic sequences.

STORM--Gary Taylor

Correlation of many weather parameters represented as stochastic sequences.

STOCHFEED--Dave Ficke

Stochastic feedback control system.

BURSTCALC--Ehud Bracha

Arithmetic operations on "burst" sequences. (A burst representation of a base r digit consists of a string of from 0 to $r-1$ ones.)

ARC (Addressable Ring Conveyor)--Kenneth Holberger

A ring communication system that, by means of automatic reconfiguration, can continue to function when various nodes are destroyed.

FM-DIRAD--Paul Mohan

All digital FM radio.

AM-DIRAD--Scott Noble

All digital AM radio.

BURSTSORT--Robert Pleva

Sorting schemes for data represented by burst sequences.

BURSTCOMM--Martin Wolff

Burst sequence encoding and decoding of video signals.

1.1 STORM (Project No. 49)

Summary

Storm is a project investigating the prediction of weather parameters, based on observed correlations between those parameters and preceding weather conditions.

1.1.2 Status of Project

Since the calculation of correlations is essentially multiplication, the encoding of the weather parameters into synchronous random pulse sequences can result in hardware economy. Designs which exploit this advantage are being considered.

Gary Taylor

1.2 MOLECULAR STOCHASTICS (Project No. 51)

1.2.1 Project Summary

Molecular Stochastics is a project concerned with the direct production of stochastic sequences from transducers. A stochastic sequence is one in which 0s and 1s occur randomly. The advantage of using these sequences is that computations (multiplication, addition, etc) are accomplished by very little hardware.

1.2.2 Project Status

In this past quarter, work has been directed toward design and construction of a stochastic meter. Its function is to accept a random signal, convert it to a stochastic sequence and determine the time average of this sequence. Figure 1 shows a block diagram of the stochastic meter. The buffer is useful for accepting various kinds of inputs and for standardizing them. The amplifier is needed to control the level of the now standardized input and the comparator converts the input signal into a stochastic sequence (RPS). The remaining blocks are for

Figure 1. Block Diagram of the Stochastic Meter

the determination of the time average of the stochastic sequence. This meter is sufficiently flexible to accept inputs from all transducers under study.

Also, the pressure transducers necessary for the velocity transducer were ordered and received. However, time did not allow for the installation of these transducers. Thus, no data on a turbulent flow is available at this time.

1.2.3 Future Work

There are three primary goals for the next quarter. First, the completion of the stochastic meter should be accomplished. Second, installation of the pressure transducer and the acquisition of data from a turbulent flow should be started. Finally, construction of a light intensity transducer should be investigated.

Jim Cutler

1.3 STOCHFEED (Project No. 53)

Stochfeed is a feedback control system whose signals are of a stochastic nature, i.e., synchronous random pulse sequences. The feedback element of Stochfeed is a transducer which captures a random component of a defined control output and produces an SRPS which is compared with a calibrated reference input. A control element which makes decisions based upon this comparison then acts through a dynamic unit in such a manner that the control output will follow the command input.

A model for Stochfeed is depicted in Figure 1. The reference input, feedback signal, actuating signal, and manipulated variable take the form of an SRPS. The dynamic unit and feedback element are of course dependent on the particular control output of interest; however, the reference selector and control element are of a more general nature.

Initially, a Stochfeed system is being investigated whose control output is the temperature of a defined environment.

Dave Ficke

R - REFERENCE INPUT
F - FEEDBACK SIGNAL
A - ACTUATING SIGNAL
M - MANIPULATED VARIABLE
C - CONTROL OUTPUT

Figure 1. Stochfeed Block Diagram

1.4 BURSTCALC (Project No. 54)

1.4.1 Introduction

The proposal (Novel PCM Transmission and Processing Systems called Burst Processing) by W. J. Poppelbaum, discusses the reasons for doing research in Burst Processing and the advantages of Burst Processing are listed, namely, high precision with simple circuits and simple circuits for arithmetic units.

In the proposal, the Block Sum Register (BSR) is defined and all the work proposed makes use of this device; at the same time, it is emphasized that conventional logic circuits must be employed as a back-up to BSR design, for comparison.

The aim of this project, BURSTCALC, is to design an arithmetic unit which accepts two Burst sequences as input and produces, with the appropriate control signals, the Burst sequence of the sum, difference, product and quotient of these.

This report discusses a conventional approach to multiplying burst sequences. It is assumed that the reader is familiar with the proposal mentioned above.

1.4.2 Digital Multiplier

In the block representation each number is represented by a block of 10 time slots and each block can contain numbers from 0 to 1, in steps of 0.1. This is illustrated in Figure 1, which shows both the number representation and a table of rounded products.

The table was examined and the logical equations for each output pulse were determined. The assumption is that the two blocks of the input Bursts are available with all their pulses in registers, and the product can be computed combinationally, and not sequentially.

The main idea is to store the two Burst sequences in two separate shift-registers and, after that is done, a logic network determines immediately the

$\begin{smallmatrix} A \\ B \end{smallmatrix}$	0	.1	.2	.3	.4	.5	.6	.7	.8	.9	1.0
0	0	0	0	0	0	0	0	0	0	0	0
.1	0	0	0	0	0	.1	.1	.1	.1	.1	.1
.2	0	0	0	.1	.1	.1	.1	.1	.2	.2	.2
.3	0	0	.1	.1	.1	.2	.2	.2	.2	.3	.3
.4	0	0	.1	.1	.2	.2	.2	.3	.3	.4	.4
.5	0	.1	.1	.2	.2	.3	.3	.4	.4	.5	.5
.6	0	.1	.1	.2	.2	.3	.4	.4	.5	.5	.6
.7	0	.1	.1	.2	.3	.4	.4	.5	.6	.6	.7
.8	0	.1	.2	.2	.3	.4	.5	.6	.6	.7	.8
.9	0	.1	.2	.3	.4	.5	.5	.6	.7	.8	.9
1.0	0	.1	.2	.3	.4	.5	.6	.7	.8	.9	1.0

Figure 1. Table of Products and Burst Notations

product pulses. These are fed in parallel to another shift register for output. Obviously, input and output can proceed simultaneously. See Figure 2.

The equations for the product pulses P1 to P10, can be derived from the Table of Figure 1. Thus, $P_{10} = 1$ if $A_{10} = 1$ and $B_{10} = 1$, i.e.

$$P_{10} = A_{10} \cdot B_{10} ;$$

similarly,

$$P_9 = A_{10}.B_9 + A_9.B_{10} .$$

The complete equations for P1 to P10 are listed below and beside them the numbers of ANDs required. Each pair i.j represents the AND $A_i.B_j$.

<u>No. of Terms</u>	<u>Equations</u>
1	$P_{10} = 10,10$
2	$P_9 = 10,9 + 9,10$
3	$P_8 = 10,8 + 9,9 + 8,10$
4	$P_7 = 10,7 + 9,8 + 8,9 + 7,10$
7	$P_6 = 10,6 + 9,7 + 8,7 + 8,8 + 7,8 + 7,9 + 6,10$
9	$P_5 = 10,5 + 9,5 + 9,6 + 8,6 + 7,7 + 6,8 + 6,9 + 5,9 + 5,10$
11	$P_4 = 10,4 + 9,4 + 8,5 + 7,5 + 7,6 + 6,6 + 6,7 + 5,7 + 5,8 + 4,9 + 4,10$
11	$P_3 = 10,3 + 9,3 + 8,4 + 7,4 + 6,5 + 5,5 + 5,6 + 4,7 + 5,8 + 3,9 + 3,10$
19	$P_2 = 10,2 + 9,3 + 8,2 + 8,3 + 7,3 + 6,3 + 6,4 + 5,3 + 5,4 + 4,4 + 4,5 +$ $4,6 + 3,5 + 3,6 + 3,7 + 3,8 + 2,8 + 2,9 + 2,10$
25	$P_1 = 10,1 + 9,1 + 8,1 + 7,1 + 7,2 + 6,1 + 6,2 + 5,1 + 5,2 + 4,2 + 4,3 +$ $3,2 + 3,3 + 3,4 + 2,3 + 2,4 + 2,5 + 2,6 + 2,7 + 1,5 + 1,6 + 1,7 +$ $1,8 + 1,9 + 1,10$

Figure 2. Simplified Logic Diagram of Multiplier

It is seen that at least 92 AND gates are needed and some OR gates too. This is not a small number of gates, but the simplicity of the proposed circuit must be compared with the usual weighted binary, for determining which method is better. This has not been done yet, but is the next step in the research.

It is quite obvious that there is a delay of one block period, between the output and the input, but this delay is not cumulative, i.e., if the input consists of consecutive blocks, then the output is so too, and one can look at the output continuously, and average over blocks and get the right result.

1.4.3 Conclusions

The use of Burst sequences in blocks of 10 pulses, as a basic representation of numbers, enables us to design a multiplier with round-off, which gives a product with an accuracy, which is better than the accuracy of the block representation itself. Still, there is more work to be done in the field, namely, comparison with other methods, examination of negative number representations, multiplication of numbers represented by super blocks, and their accuracy, and design of a divider.

Ehud Bracha

1.5 ARC (Project No. 56)

1.5.1 Introduction

The Addressable Ring Conveyor system is a proposed damage tolerant information network for shipboard communication. A major difference between ARC and other communication systems is in how it recovers and reconfigures itself when random lines are severed and/or random subsystems are destroyed. All other networks use software tests to deduce the needed changes. Furthermore, in non-loop networks, most systems require that each station be linked to at least two

others to survive just one hit. The ARC system is a double loop network that uses trivial hardware tests for reconfiguration in the case of one hit or multiple adjacent hits. If two central points (not exchanges) are added, continuity can be maintained in the event of two groups of one or more adjacent hits.

1.5.2. General Description

ARC will be a synchronous, byte multiplexed network with addressed blocks. Information will circulate around the ring in frames, passing through each station once for each circumnavigation. The stations behave like 16-bit shift registers. The Frames are shifted out the end of one station into the beginning of the next.

Figure 1 shows the frame format that is currently being considered, Bipolar modulation is used. This signal can be separated into two registers shifting in parallel and recombined with trivial asynchronous circuitry. The positive pulses carry the 10-bit information field, N_i , and the 5-bit instruction field. The negative going pulses contain the 7-bit address. An address of 0000000 will denote a 'null' or empty frame. The address 1111111 will not be allowed because it would be indistinguishable from the monitor. This leaves 126 possible addresses. The monitor field always contains seven ones. It is used for synchronization in the stations and is one of the criteria for determining if the stations are damaged or not.

To execute a read operation, a station will gate the information field and the instruction field into a buffer and replace the frame with a null frame. To execute a write operation, a station will gate the information, instruction, and address fields into a passing null frame. If the number of non-null frames on the ring approaches the capacity of the system, the capacity can be increased by switching extra shift registers into the ring. When the traffic dies down,

Figure 1. Frame Format

these dummy shift registers can be removed. In a like manner, if certain stations need a higher message rate, stations that are not in use can be switched out of the ring.

1.5.3 Damage Tolerance

By looking continuously at the monitor and fixed zero bits as they come out of each station, a crude judgment can be made of the stations' health. A signal from a station's monitor circuit controls switches called commutators at each end of the section of the ring containing that station. Refer to Figure 2. The commutators are wired such that the absence of a signal isolates the station and routes the data on the inner ring from the previous station through the outer ring to the following station. The data does not pass through the damaged area. Adjacent sections may be eliminated without disrupting the continuity of the healthy ones. Figure 3 shows a possible realization of the monitor-commutator circuit.

By combining the power and clock supplies in a separate single ring (with appropriate fuses), all stations are assured power and synchronizing signals. Figure 4 shows a possible realization of the power and clock circuits.

Figure 5 shows how two groups of one or more hits can be survived with the addition of a zenith point, Z, and a nadir point, N. A second hit would be bypassed by leaving the "plane of the ring" and routing data into a 3D arrangement of "spokes" via N and Z. It is also easy to assess that this situation exists: If a sector already has information flow in the emergency loop, then the next step is to go via N and Z rather than split into two smaller, isolated rings.

1.5.4 Project Status

A study of general communication networks, information formatting in specific networks, general packet switched networks, and specific loop networks

EACH STATION CONTROLS (VIA ITS "SHIFT-MONITOR," SEE BELOW) TWO COMMUTATORS: NORMALLY THESE LEAVE THE INNER AND OUTER RINGS SEPARATE. FAILURE OF A STATION TO OPERATE PROPERLY ELIMINATES IT WITH ITS SECTOR AND USES THE OUTER RING FOR CONTINUITY: THE SIGNALS NOW DO NOT PASS THROUGH THE ENDANGERED SECTOR! NOTE THAT ADJACENT SECTIONS CAN BE DESTROYED TOGETHER. DESTRUCTION OF A PART OF THE POWER/CLOCK RING IS IMMATERIAL; FUSES ELIMINATE THE SHORT-CIRCUITED SECTIONS...

Figure 2. Surviving a Single Hit

Figure 3. Monitor and Commutators.

NOTES:

1. THIS IDEA IS FEASIBLE BECAUSE OF THE SMALL POWER CONSUMPTION (<1W/STATION)
2. FILTERING IS TRIVIAL AT 1MHz (EVEN TRANSFORMERS ARE USABLE IN MINIATURE FORM)

Figure 4. Power Clock for ARC System

NOTE: BY SENSING (AT THE END OF EACH SECTION) WHETHER INFORMATION IS ALREADY PRESENT ON THE EMERGENCY RING, ONE CAN GO AROUND A SECOND HIT VIA THE ZENITH POINT Z AND THE NADIR POINT N. THE DIRECTION OF INFORMATION FLOW IN THE REMAINING STATIONS IS UNALTERED.

Figure 5. Surviving Two Arbitrary Hits (3D-Extension)

is nearly complete. Different and larger frame formats are being investigated. A faster and more rigorous health assessment and commutator circuit has been proposed and is being designed. A power extractor circuit that will supply more current without dropping the voltage level is also under study.

1.5.5 Future Work

During the next quarter, a prototype monitor-commutator circuit will be built and tested. The read and write operations will be studied and realization of the stations will be started. The possibility of making the stations more than one frame length long to ease the synchronous insertion and deletion of dummy frames will also be studied.

Kenneth Holberger

1.6 FM-DIRAD (FM Digital Radio) (Project No. 59)

1.6.1 Introduction

With the widespread use of digital circuits, it is inevitable that digital techniques should find their way into radio receiver design. The desirability of such a change is found in the qualities generally associated with digital circuitry: reliability, low cost and small size. Thus, it is the intent of this project to produce one such digital receiver which will operate in the FM broadcast band and to compare its performance with that of present analog receivers.

1.6.2 Scope of Work

Indications of the trend towards producing radio receivers composed mainly of digital building blocks have appeared in much of the literature over the past ten years. The advent of fast Fourier transforms and other digital signal processing techniques has made the digital detection problem

much easier to handle. However, many of the publications concerning possible digital receivers have been analytical in nature, stressing the mathematical conceivability of such devices. The goal of this research is to devise a working model of an FM digital receiver. Although the analytical concerns of digital signal processing are of utmost importance, further development of this area will not be stressed. Rather, the implementation of some of the ideas already present, along with some new techniques which appear appropriate, will be our major concern. Once a prototype has been constructed we can then proceed to observe the relative merits of this particular receiver and initiate any design modifications which would seem to better the performance of the device. Therefore, hardware orientation, guided by the analytical studies conducted by others, will be the underlying theme of this research.

1.6.3 Initial Ideas

The general form of a digital receiver is depicted in Figure 1. Here, the input is a frequency modulated carrier, either direct from the RF amplifier or out of the IF stage of a conventional receiver front end. The input is then appropriately sampled and quantized, possibly by an A/D converter. The digital data is then processed somehow and the information impressed upon the carrier is extracted.

Figure 2 shows one variation of the general form of digital receiver which is well suited to FM reception. In this case the input will be the modulated RF with carrier frequencies of about 100 MHz. This signal is passed through a phase-locked loop which serves as a frequency selective block and also tends to filter out wideband noise. The VCO output of the phase-locked loop is passed through a zero-crossing detector and the resulting train of impulses of modulated period is then compared with a series of impulses of fixed period (perhaps the period of the carrier). Since the change in period

Figure 1. General Digital Receiver

Figure 2. FM Digital Receiver

of the modulated impulses is very small, the method of comparison of the two will exhibit properties similar to those of Moiré patterns. The Moiré pattern technique has the ability of producing low frequency "beats" when two impulse trains of closely matched periods are multiplied. This multiplication process need not be analog multiplication, however, but may simply be the ANDing of the modulated and fixed period impulses. The low frequency "beats" produced by this Moiré detector would then represent the information sent on the carrier. This relatively simple concept of FM detection appears promising in that it employs inherently digital techniques. The particulars of the detection scheme are yet to be determined as well as the performance capabilities of this type of detector. It is hoped that this particular type of digital receiver can be breadboarded and refined to the point where it will at least indicate that such a receiver is advantageous and warrants further circuit design in this field.

Paul L. Mohan

1.7 AM-DIRAD (Project No. 60)

1.7.1 Introduction

The purpose of this new project is to study the feasibility of applying digital signal processing techniques to various portions of a general purpose AM receiver. The hoped for results are increased reliability, lower unit cost, and better performance.

1.7.2 Preliminary Considerations

There is a question as to how much of a radio should be made digital. The front end should probably be analog for good sensitivity. Since any high frequency (> 10 MHz) would need extremely fast circuitry to keep up with the changing waveform, the best place to start digitizing would be in the IF,

around 1 MHz. Since 455 kHz is a common AM-type IF frequency, a preliminary digital detector will be developed there.

Scott Noble

1.8 BURSTSORT (Project No. 62)

1.8.1 Introduction

Projects are currently underway to assess the performance of the "BURST" PCM system when applied to practical processing problems in environments where noise and damage tolerance are crucial. Arithmetic functions are obviously important in most data processing systems, and it may be shown that the burst encoding is particularly amenable to such operations. Equally important, however, is the ability to perform logical operations upon numeric data items; i.e., to be able to make decisions based upon the value of received data. An obvious application of this idea can be found in the processing of radar data--here the ability to sort data makes possible such functions as range gating, removal of uninteresting "clutter," and signal enhancement through averaging. Project BURSTSORT is an investigation into the hardware implementation of high speed data sorting operations in a burst-encoded PCM data network.

The notion of "sorting" data items can take on any of several interpretations--the sort may be performed on the value of the data only, on a "key" field which identifies the data item, or on a combination of these two modes. Generalized sorting capability requires that each of these functions be provided for. Furthermore, all possible numerical relations (equality, lesser magnitude, greater magnitude, etc.) may prove necessary in various applications and must be available. Finally, once the fulfillment of the sorting criteria has been determined, any of several system responses may be required--e.g., rerouting of the original burst to a specified location, recording the

event in a memory circuit, simple "go/no-go" response on a given line, or even real-time modification of the sort criteria in the case of an "adaptive" system. Of course, any "sortable" burst sequence must be of finite length: therefore the existence of some "end-of-frame" delimiter is assumed.

At first view, burst encoding, as a variant of the general PCM method, seems naturally suited to processing in the digital mode. After all, a burst value can be converted to a binary value by the simple expedient of counting pulses during a fixed time period. Counting, however, can be a relatively slow process due to the ever-present problem of carry propagation. An alternative is found in the block sum register, or BSR, which provides analog representations of the burst data and which is not limited in its speed of operation by carry problems. The high speed, simplicity, and projected low cost (in integrated form) of the BSR approach make it a preferred alternative to strictly digital processing.

1.8.2 Project Status

It can be seen, then, that burst sorting in the analog mode requires three basic components: one or more block sum registers to decode the data, a source of reference voltage(s) representing the desired sorting criteria, and analog comparison circuitry. A general system outline is depicted in Figure 1. The use of the shift register (BSR) technique as opposed to counting pulses provides an extra bonus: the entire burst is present exactly as received, thus there is no need to regenerate the sequence should retransmission be required. The limitations imposed by the system throughput requirements and analog circuitry response times may make it necessary that the data registers (BSR's) be duplicated and interleaved (or perhaps even serially connected in "pipeline" fashion) so as to relax speed restrictions. The problem of reference

Figure 1. Overview of BURTSORT

voltage generation (as determined by the KEY field) is of prime importance-- a possible approach would be to utilize a high speed read-only memory in conjunction with digital to analog conversion circuitry. The effectiveness of such a technique has yet to be fully investigated.

1.8.3 Future Work

The detailed features of the BURSTSORT system have yet to be finalized. The problem of implementation must then be fully analyzed, and the design of an operational prototype will then begin. One possibility is that the device will be made compatible with the format of the "ARC" data transmission system currently being developed. This would enable a more complete evaluation of the performance of the burst mode in a combined data transmission/processing system.

Robert M. Pleva

1.9 BURSTCOMM (Project No. 63)

1.9.1 Summary

BURSTCOMM (Burst Communications) is a project involving the encoding and decoding of a video signal using burst techniques. With a BSR (Block Sum Register) and a comparator as a staircase encoder, and a BSR as a decoder, a simple low cost digital video transmission system is possible.

1.9.2 Progress

Initial experiments transmitting low frequency sine waves using Al Irwin's audio frequency encoder suggest a minimum clock frequency of 40 times the signal frequency to give clear results. Therefore the initial video encoding clock is at 40 MHz giving a probable clear transmission of video signals of at least 1 MHz. A simple filter will be used to limit the incoming video signal to a resolution of 1 MHz.

Due to the high frequencies involved it was decided to use ECL 10,000 series logic. Schottky TTL would suffice for 40 MHz, but it may turn out that for better video resolution higher clock frequencies are needed. Recent price decreases in ECL 10,000 series chips have made using them much more economical.

So far, one BSR using ECL D flipflops and high frequency transistors (2N 4257a) has been built. Significant noise (at 40 MHz) at the stairstep output has still to be eliminated. In the future, the following items will be investigated: what minimum clock frequency will still give a usable video output, and what minimum number of steps in the stairstep at various clock frequencies will give a useful output.

Martin Wolff

II. HARDWARE RESEARCH (VARIOUS)

Summary

The reports in this section are as follows:

FROG--Dev Bose

Adaptive robot that learns from visual clues.

CAECOTRON--Bernard Tse

An aid to the blind, converts special stereo TV images into audible tones.

THESPIAC--Les Daley

Theatre lighting control computer. Prototype is now complete and detailed report will follow.

NORMAN (Normalizing Analyzer)--Garlan Huberts

Identifies certain plane figures regardless of orientation.

NASACOMP--William Stenzel

An array computer, after Illiac IV.

CARDALERT--S. K. Yuen

An automatic hardware cardiac monitor for detecting irregularities in an EKG.

2.1 FROG (Project No. 36)

2.1.1 Introduction

FROG is an autonomous mini-robot which learns to sustain its existence in its environment. It uses visual clues to learn to distinguish between food and potential source of danger. This it does by building a model of the external world within itself.

2.1.2 Project Status

In this quarter primary attention was focused on devising a simple and reliable control mechanism for the movable block (MB). Our initial idea of maintaining the block on course between two hidden parallel copper lines by inducing voltage spikes on them from an on-board oscillator (cf. QTPR April-June, 1974) had to be discarded. This is because the noise generated by the motors completely masked the control signals on the copper lines. At the present moment, we control MB by making it follow illuminated paths that are detected by photocells.

2.1.3 Future Work

Attention will now be turned towards completing the design of the world model. The structure of the world model is quite modular, the principal building block being a T*-element. A hardware implementation of a T*-element requires about 25 IC packages. Since we need 42 T*-elements, we seem to be talking in terms of 1000 IC packages or more. At this point we feel that some amount of multiplexing may save a lot of hardware. In other words, if we build one T*-element, with sufficient memory we can time multiplex this element to simulate the 40 or more we need. This is what we plan to look into next.

Dev Bose

2.2 CAECOTRON (Project No. 43)

During the last quarter, work was done on the processor of the system to try to determine depth information from the video signals generated by the two television cameras. The system is now capable of a full 16 x 16 resolution even when one of the television pictures is shifted up to four positions during the depth determination process, as described in earlier reports. This would mean that the system can retain its full resolving power when the object in question is as little as two feet in front of the user.

In the next quarter, further work will be done on the interfacing of the processor with the rest of the system. The information generated by the picture matrix generators will be put in the correct format so that the processor can act on them efficiently. Further work will be done on the tone generator circuits where two integrated circuits will be used to replace the one that is used currently.

Bernard K. Tse

2.3 THESPIAC (Project No. 47)

This quarter's efforts were directed toward completion of the hardware phase of the project with the result that the system became operational during the last week of September. The basic control programs (which were written during the design phase) were then tested. Several routines were found to misbehave when presented with boundary cases; the offending routines were modified to detect and handle these special cases.

At this time, the hardware is believed to be completed and the basic control programs are now debugged. Assuming that no changes are requested by lighting designers, the software is also complete. The following two quarters will be directed toward the preparation of a final project report in the form of a thesis.

2.4 NORMAN (NORMAlizing ANalyzer) (Project No. 50)

During this latest quarter, all of the design work was completed on NORMAN. Only construction and debugging remain to be done.

The Switching Point Controller and Stochastic Sequence Generator boards mentioned in the last quarterly report on this project have since been debugged. Also, ten boards which contain the memory of the machine were designed and constructed. Each pattern is stored on a set of two boards so that five patterns can be stored in the machine. Pattern storage takes two boards because quite a bit of processing has to also be done. Storage is in a RAM type memory (SN7489) which I term a "profile buffer" because it contains the normalized profile of the input figure. This buffer is arranged as eleven words of twelve bits each. During the storage mode, each word is simply filled with a number from a counter. In the analyze mode (when one is asking the machine to "recognize" a pattern) the machine has to find the absolute difference for each word of the memory and the aforementioned counter. It also has to sum all eleven of these absolute differences together to obtain a final number called a "fit." This number is a measure of how closely the profiles of the two figures match.

Near the end of the analyze cycle the machine has six fit numbers (five storage locations and one threshold) to compare. The comparisons are done by a board called the Result Analyzer board which finds the lowest of the six numbers and sends it on to the Display board. Also sent on is the name (triangle, etc.) attached to this number. The Threshold board simply acts as a storage location which contains a settable fit number. If this number is the lowest of the six, the machine will simply say in effect that it doesn't recognize the figure. All these boards have been built, but most have yet to be checked and debugged.

Garlan Huberts

2.5 NASACOMP (Project No. 57)

2.5.1 Overview

The aim of this research is the design of a large, high speed, digital, parallel processor with ILLIAC IV style architecture. It is to consist of an array of arithmetic processing elements, each with an independent data memory of 4K to 8K 32-bit words. These elements (PE's) are to execute a common instruction stream under the control of a central control unit (CU). The control unit is to have its own independent instruction memory.

The processors communicate data with one another by means of a network of crossbar switches. Each processor is uniquely connected to one port of a crossbar switch by means of a bi-directional data bus. The crossbars are further connected to each other by means of bi-directional busses in a configuration known as the 'perfect shuffle interconnection.' The aggregate of crossbars and their interconnections is referred to as the 'routing network'; each crossbar with its concomitant bus drivers, receivers, etc. is called a routing box. The routing network is to be under the control of the CU.

The remainder of this report is chiefly concerned with the processing element (PE) and its high speed parallel multiplier. Figure 1 shows the general structure of the machine.

2.5.2 Arithmetic Processing Element Design

The design has an IBM 360 single precision 32 bit floating point number format - that is, a 1 bit sign, a 7 bit base 16 excess 64 exponent and a 24 bit mantissa.

The operations which have, at this point, been incorporated into the design include a floating point multiply, add and subtract. The execution time for these operations is to be somewhat less than one memory cycle (tentatively 500 - 600 nsec).

Figure 1. Global Structure

The multiply operation involves a novel multiplication scheme utilizing a large number of read only memories. Thirty-six 256 by 8 bit ROMs, each programmed to produce the 8 bit product of two 4 bit operands, are used in parallel to generate a 48 bit wide matrix of partial products. (One partial product is generated for each ordered pair of 4 bit segments in the Cartesian product of the segments of the multiplier and the multiplicand. The operands are 24 bits, hence there are six 4 bit segments and 36 pairs.) This matrix of partial products is then reduced, principally by 1024 by 4 bit ROMs, to give a 48 bit wide matrix with only two rows. The reduced matrix may then be reduced to one 48 bit product by means of 4 bit slice arithmetic logic units (ALUs) employing carry look ahead. This scheme is similar in concept to Wallace's or Dadda's multiplication scheme, but represents utilization of considerably fewer IC packages.

All operations tentatively provide for 2 extra hexadecimal 'guard digits' in their result, which may be used to increase the accuracy of subsequent adds and subtracts or which may be rounded to give a 24 bit result. A file of eight or more 40 bit registers will be provided to allow for temporary storage of the result with guard digits. The result will be shortened to 32 bits before storage in the memory.

Figure 2 contains a rough sketch of the primary data paths in a processor. Omissions have been made for the sake of simplicity--among the omissions are the processor's 24 by 24 bit multiplier, Read Only Memory used for table lookup in division and hardware for exponent accumulation. Also omitted are most of the external control and clocking signals.

The sketch shown is roughly separable into three sections: a section for prealignment of operands in floating point add and subtract operations, an accumulator section containing the arithmetic-logic unit and registers and a

Figure 2. Block Diagram of a P.E.

section for post normalization of floating point results. These sections correspond to the steps in performing a floating point add/subtract. A description of the data flow and some of the internal control for these steps follows.

2.5.3 Prealignment

Before performing an add/subtract, one of the operands must be shifted to the right until the exponents of the two operands agree. The processor does this as follows (starting at the top of the sketch). The A exponent minus the B exponent is computed as a 2's complement number, and both operands are inspected for a mantissa of zero. This information is input to the prealignment control, which determines which of the operands (if either) is to be shifted, and by what amount. Both the A and B operands enter identical shift right networks (prealign shift on the diagram). Each shift network consists of four asynchronous 8 position scalars or 'barrel switches'. The amount of the shift is, of course, determined by the prealign control, which in this case generates the proper shift amount for the shifted operand and a zero shift amount for the unshifted operand.

2.5.4 Accumulator

Data enters the accumulator section through 2 banks of 2 to 1 selectors (one for each operand), which select either the prealignment section or the processor's multiplier as the data source. The signs of the operands and the instructed operation are examined by the ALU function control to determine the actual operation which the ALU is to perform, and to generate a sign for the result. e.g. if the instruction specifies 'add' with A being positive and B negative, the actual operation which the ALU performs is a subtraction, and the sign of the result is dependent upon relative magnitudes of the operands.)

2.5.5 Normalization

The result of the operation must now be post-normalized, i.e., shifted to the left to remove leading zeros or shifted to the right in the case of an overflow. The accumulator contents, as well as a control signal indicating overflow, are inspected by the normalization control, which specifies a shift amount and direction (right or left) to the normalization shift network. The normalization shifter consists of an independent right shifter and left shifter, each of which is composed of four 'barrel switches' similar to the prealign shifters. The output of the normalization network is routed back to the top of the accumulator section, where it may be rounded to remove the guard digits which are used in performing operations. (Since the barrel switches in both the prealign and normalization networks have open collector outputs, the routing back into the accumulator section may be accomplished by wiring the outputs of the normalization net and one of the prealign nets together, and insuring that one or the other is always disabled. Hence, this connection is shown as a wire-AND.) It is possible when rounding that an overflow will occur, hence provision is made to shift right, re-normalizing the result, after which the result flows through the accumulator section, where it is clocked into the register.

2.5.6 Prototypes

Efforts are currently underway for the construction of a prototype 24 x 24 bit multiplier and of a reduced version of the routing network. It is expected that a prototype processor will eventually be constructed but this has been deferred in favor of the multiplier and routing network.

The physical partitioning of a PE will require data flow between the elements of the PE, e.g., data in and out of the multiplier or memory. Traditional

backplane interboard connection schemes route the data to metallized "fingers" at the edge of a board, which then plug into an "edge connector" in the card cage. The boards are interconnected by means of wire-wrapped connections between the corresponding edge connectors.

The large number of data paths between boards would require a correspondingly large number of fingers and wire-wrapped connections. Also, layout of boards using this approach is complicated by requiring that the data enter and exit on the same edge of the board. In view of these facts, we have been examining interconnection with flexible ribbon cable and plug-in connectors. The flexible cable approach allows greater freedom in board layout (inputs and outputs need not be at one edge of the board) and its assembly is much simpler than wire-wrapping. (Plugs at the end of the cable are crimped on in one operation regardless of the number of lines in the cable, and sockets on the boards use standard IC pin spacing.) There are a variety of vendors in this area, including 3M and AMP.

The board layout for the multiplier is well underway, and after several iterations in package placement schemes, it appears that the entire 24 x 24 bit multiplier will occupy one double sided PC board of approximately 15" x 15". The question of an adequate power distribution system arises when dealing with high density boards (such as our multiplier). We will use a bus bar-type arrangement for both power and ground return. This arrangement will have a low profile, and hopefully will not affect interboard spacing.

Parts have been ordered for a routing network of four routing boxes. Each of them will constitute a 4 x 4 crossbar switch for a 4-bit wide data path. We are still awaiting delivery of these parts as well as the ROM's and adders for the multiplier.

William Stenzel

2.6 CARDALERT--An Arrhythmia Monitor (Project No. 64)

2.6.1 Introduction

The goal of the CARDALERT project is to implement with modern technology a portable electronic system which monitors arrhythmic behavior of the human heart.

A normal EKG diagram is shown in Figure 1, which consists of a series of upward and downward deflections. Alphabets are designated to the peaks which actually represent the different activities of the heart. Peak P corresponds to the contraction (depolarization) of the atria; QRS, the contraction of the ventricles; and T, the recovery (repolarization) of the ventricles. Figure 2 is used to demonstrate these relations. Note that the average time durations between the peaks are also given in the figure. During abnormal activities of the heart, the normal rhythm of the heart pulses is interrupted. Consequently, the time relations of the deflections are changed or the shape of the waves is distorted, or both.

2.6.2 Specifications of CARDALERT

CARDALERT is designed to detect various arrhythmias. The technique is to detect the time intervals between various peaks and test if they are abnormal. Any single abnormality or cumulative occurrence of these abnormalities can cause an alarm to warn the patient of arrhythmia activities. Table 1 outlines the alarm conditions.

2.6.3 CARDALERT Block Diagram

Figure 3 shows the simplified block diagram of the CARDALERT system. The cardiac signal sensed by a pair of electrodes is amplified by a high gain differential amplifier with a very high CMRR. The signal is then filtered and changed into digital values at 250 samples per second. Leaving the Data

Figure 1. Normal EKG

Figure 2. Relationship of EKG to the Heart

ABNORMALITY	LIMIT	NO. OF OCCURRENCES TO CAUSE AN ALARM
Low Rate	< 50 bpm	10 or more per min.
	< 40 bpm	1
High Rate	> 120 bpm	6 consecutive
Short PR	< 0.12 s	12 or more per min.
Long PR	> 0.21 s	5 or more in 15 beats
Aberrancy	QRS > 0.12 s	4 consecutive
Rhythm	RR intervals	
Changes	vary by $\pm 20\%$ from preceding RR interval	8 or more per min.
Premature	RR intervals occurring	
Beats with	within 80% of the	2 consecutive or
Wide QRS	preceding RR interval	6 per min.
	with QRS > 0.12 s	
Premature Beats	RR intervals occurring	
with Compensatory	within 80% of the	2 consecutive or
Pause	preceding RR interval	6 per min.
	followed by a pause	
	> 125% of the average	
	RR interval	
Absent	P not found within	12 or more per min.
P Wave	0.35 preceding the QRS	

Table 1. CARDALERT ALARM CONDITIONS

Figure 3. Simplified CARDALERT Block Diagram

Acquisition block, the digitized signal is first processed in the Preprocessor block from where relevant data is extracted before further processing in the time multiplexed main processor unit.

2.6.4 The Preprocessor

Essentially, the Preprocessor (Figure 4) performs the digital differentiation of the incoming signals. The slope (given in terms of the difference between the immediate past and the present value) is then compared with a slope threshold. Only if the slope is bigger than the threshold is the candidate counter incremented. At least three such consecutive candidates have to be recorded in order to satisfy the lower bound requirement. An R peak is then considered to have occurred. An RR counter counting at 1 KHz is then triggered as a result. Its content is buffered into Register 5 at the next R peak and the counter cleared to start from zero again.

The sequences of positive and negative signs from the digital differentiator are counted in the '+' sign counter and the '-' sign counter, respectively. At the detection of a QRS occurrence, the present contents of the sign counters are multiplexed by 4 giving PQ and QR intervals in milliseconds. The immediate following content of the '+' sign counter is multiplexed by 4 again to give the RS value. All these intervals, namely: PQ, RS, QR are appropriately added together to give the PR and QRS intervals. The RR, PR, QRS intervals are further processed in the time multiplexed main processor to test for arrhythmia situations.

S. K. Yuen

Figure 4. The Preprocessor

3 SOFTWARE SYSTEMS RESEARCH

(Supported in part by the Atomic Energy Commission under grant No. US AEC AT11-1(2383))

3.1 Numerical Processes

3.1.1 Tests of Integration Packages (L. Petzold, C. W. Gear)

The program ZKUS4 for stiff systems, submitted to C. W. Gear for evaluation, was tested and compared with DIFSUB. Both programs were used to integrate the system

$$y_1' = 10^4 y_2 y_3 - 4 \times 10^{-2} y_1,$$

$$y_2' = 4 \times 10^{-2} y_1 - 10^4 y_2 y_3 - 3 \times 10^7 y_2^2,$$

$$y_3' = 3 \times 10^7 y_2^2.$$

$$y_1(0) = 1, y_2(0) = y_3(0) = 0,$$

over the interval $[0,4]$ in single and double precision. The programs produced significantly different values for y_2 near $t = 0$. To better examine this difference, y_2 was scaled by a factor of 10^4 and a new equation $y_4' = y_2$ was added to the system. The programs produced significantly different values of y_4 at $t = 4$, indicating a substantial difference between the values of y_2 produced over the interval $[0,4]$. However, ZKUS4 used less computer time, so it could not be rejected outright.

Currently NEWDIF, a new experimental version of DIFSUB, is being studied and tested on simple linear systems. Hopefully these studies will lead to modifications increasing the efficiency and power of DIFSUB.

3.1.2 Periodic Solutions of Ordinary Differential Equations (M. Schweitzer, C. W. Gear)

Our past approach has been to approximate the periodic solution by a linear combination of splines with fixed, equally spaced knots. Unfortunately,

this approach has proved unsatisfactory, requiring large numbers of knots for acceptable accuracy. Hopefully, the number of knots can be reduced by allowing the knots to vary.

If we approximate the solution of $\underline{y}' = \underline{f}(\underline{y}, t)$ by

$$\underline{z}(t) = \sum_{i=1}^n \alpha_i \phi_i(t),$$

where the ϕ_i 's form a spline basis with knots t_i , and $[0, t_n]$ is the period, one plausible way of choosing the coefficients α is to minimize the integral

$$I = \int_0^t n [\underline{z}'(t) - \underline{f}(\underline{z}(t), t)]^2 dt.$$

This requires that

$$\partial I / \partial \alpha_i = 0, \quad \partial I / \partial t_i = 0, \quad i = 1, 2, \dots, n.$$

If the system $\underline{y}' = \underline{f}$ contains m equations, then a system of $(m+1)n$ nonlinear equations must be solved. Since our previous approach required the solution of a system of mn equations, the new approach does not require significantly more work. It also has the added advantage for autonomous systems of permitting the period to vary. This new approach is being tested for linear systems using linear splines.

3.1.3 Analysis of Methods for Ordinary Differential Equations (R. Skeel)

A unified approach for analyzing nearly every constant-stepsize constant-order method has been developed. The new theory is helpful in two ways:

- 1) as a pedagogical tool. The same elegant and transparent approach used by Henrici (and most other authors) for one-step methods is now applicable to "multistep" methods.
- 2) as a research tool. The difficult problem of global error analysis is reduced to the simpler problem of analyzing the local error.

It has been known for a while that the concept of consistency is inadequate for many methods which are not of the form

$$y_n = \alpha_1 y_{n-1} + \dots + \alpha_k y_{n-k} + h\psi(t_{n-1}, y_{n-1}, \dots, y_{n-k}, h).$$

The appropriate concept, termed "quasi-consistency", has been found, and it has been applied to linear Nordsieck methods to obtain a simple condition for optimal order,

$$B_1 \ell_1 + B_2 \ell_2 + \dots + B_k \ell_k = 0,$$

where the B_i 's are the Bernoulli numbers and the ℓ_i 's are the components of the corrector vector.

Conditions on the variation of the stepsizes sufficient to guarantee the stability of methods using the interpolation step changing technique have also been developed. For example, for strongly stable methods, it is sufficient to require that the step changing ratios $\omega_n = h_n/h_{n-1}$ lie in a set of the form $[0, \alpha][\beta, \gamma]$, where $0 < \alpha \leq \beta < 1 \leq \gamma$. Thus drastic reductions in the stepsize need not jeopardize stability.

The global error for variable stepsize methods was also studied and the asymptotic form of the global error for these methods has been determined.

3.1.4 Absolute Stability of Multistep Methods (A. Kong, R. Skeel)

The regions of absolute stability for two general multistep methods were investigated using programs designed to plot the regions on a CRT. The first method has the form

$$\begin{aligned} (1-b)\{ \nabla y_n - (1-a)h \sum_{m=0}^{k-1} \gamma_m \nabla^m f_{n-1} - ah \sum_{m=0}^k \gamma_m^* \nabla^m f_n \} \\ + b(k+1)\gamma_{k+1}^* h \frac{\partial f}{\partial y}(t_n, y_n) \{ \sum_{m=1}^k \frac{1}{m} \nabla^m y_n - hf_n \} = 0, \end{aligned}$$

where

$$\gamma_0 = 1,$$

$$\gamma_m = 1 - \frac{1}{2} \gamma_{m-1} - \dots - \frac{1}{m+1} \gamma_0, \quad m = 1, 2, \dots$$

$$\gamma_0^* = 1$$

$$\gamma_m^* = -\frac{1}{2} \gamma_{m-1}^* - \dots - \frac{1}{m+1} \gamma_0^*, \quad m = 1, 2, \dots$$

This method includes as special cases

- 1) the Adams-Bashforth method of order k ($a=0, b=0$),
- 2) the Adams-Moulton method of order $k+1$ ($a=1, b=0$),
- 3) the backward differentiation method of order k ($b=1$),
- 4) Enright's method of order $k+2$ ($a=1, b=1/2$).

The equation for the boundary of the region of absolute stability in the complex $\mu = h\lambda$ plane is

$$A\mu^2 + B\mu + C = 0,$$

where

$$A = -b(k+1)\gamma_{k+1}^*$$

$$B = (b-1)\left\{(1-a) \sum_{m=0}^{k-1} \gamma_m \zeta^m (1-\zeta) + a \sum_{m=0}^k \gamma_m^* \zeta^m\right\} \\ + b(k+1)\gamma_{k+1}^* \sum_{m=1}^k \frac{1}{m} \zeta^m,$$

$$C = (1-b)\zeta,$$

$$\zeta = 1 - e^{i\theta}, \quad 0 \leq \theta \leq 2\pi.$$

For given values of k and a , the values of b yielding the largest angles α for which the method is $A(\alpha)$ -stable were determined experimentally. The value $a = 1$ appeared to maximize the α 's. The following table summarizes the results.

<u>k</u>	<u>b</u>	<u>α</u>
2	[.4,1]	$\pi/2$
3	[.486,1]	$\pi/2$
4	[.521,1]	$\pi/2$
5	[.536,1]	$\pi/2$
6	[.541,.738]	$\pi/2$
7	[.541,.591]	$\pi/2$
8	.540	$\tan^{-1}(10.6037)$
9	.528	$\tan^{-1}(1.78055)$
10	.525	$\tan^{-1}(1.09648)$
11	.522	$\tan^{-1}(0.54724)$

No promising results were obtained for $k \geq 12$. It appears that the maximum order for which the method is stiffly stable is 12. Hence the method compares favorably with the backward differentiation methods and Enright's methods for which the maximum orders are 6 and 9. However, the present method requires more computation than these methods.

The second method has the form

$$\sum_{j=1}^k \delta_j \nabla^j y_n = h \sum_{u=0}^k \gamma_u \nabla^u f_n,$$

where

$$\delta_1 t + \delta_2 t^2 + \delta_3 t^3 + \dots$$

$$= \{1 + at + \frac{a(a+1)}{2!} t^2 + \dots\} \{t + \frac{b-a+1}{2!} t^2 + \frac{(b-a+1)(b-a+2)}{3!} t^3 + \dots\},$$

$$\gamma_m + \frac{1}{2} \gamma_{m-1} + \dots + \frac{1}{m+1} \gamma_0 = \delta_{m+1}, \quad m = 0, 1, 2, \dots$$

The method is derived from the integration formula

$$\frac{y(b)-y(a)}{b-a} = \frac{1}{b-a} \int_a^b f(t, y(t)) dt,$$

and includes special cases

- 1) the Adams-Moulton method of order $k+1$ ($a=-1, b=0$)
- 2) the backward differentiation method of order k ($a=0, b=0$)

3) the Milne-Simpson method of order $k+1$ ($a=-2$, $b=0$).

The equation for the boundary of the region of absolute stability in the complex μ plane is

$$\mu = \frac{\sum_{j=1}^k \delta_j (1-e^{-i\theta})^j}{\sum_{j=0}^k \gamma_j (1-e^{-i\theta})^j}, \quad 0 \leq \theta \leq 2\pi.$$

As before, the values of the parameters were varied in an attempt to find new formulas, but no promising results were obtained.

3.1.5 DIFSUB and Partial Differential Equations (M. Ostrar, D. Watanabe)

Several new stretching transformations were applied to Burger's equation. As before the results were mixed. A typical transformation significantly increases the number of function evaluations required for the integration, and while it often greatly increases the accuracy over parts of the interval of integration, it also worsens it over other parts.

The effect of the transformations on the stiffness of the systems of differential equations generated by the method of lines was also investigated. In every case examined, the transformation increased the stiffness. This fact may explain the increase in the number of function evaluations required for the integration.

An investigation of the general properties of the method of lines was also begun. Programs are being written to study the variation of the global error for Burger's equation as a function of the spatial mesh and the difference approximations to the spatial derivatives. This study is similar to that performed by Loeb and Schiesser for the heat equation. The preliminary results are consistent with their results.

3.1.6 Global Error for Ordinary Differential Equations (B. Link, D. Watanabe)

Work continued on experimental and theoretical studies of the global error for ordinary differential equations.

The experimental studies involved three integration codes: (1) DIFSUB, the standard library version, (2) BDF1, an experimental version of DIFSUB, designed for ease of modification, and (3) BDF2, a code which uses backward differences in the manner of Krogh. These codes were to be modified where necessary to control the local error so that either local error/unit step $\leq \text{EPS}$ or local error/step $\leq \text{EPS}^{1+1/p}$, where EPS is the error tolerance and p is the order of the method used for the step. However, BDF1 and BDF2 behaved badly, driving the stepsize to zero for moderate values of EPS, and were temporarily set aside. DIFSUB worked reasonably well, but its error behavior was erratic. Figure 1 is typical of the results obtained when DIFSUB attempted to set the local error/step $\leq \text{EPS}$.

Simple fixed order methods were studied to analyze the effect on the global error expansion of estimating only the lower order terms of the local error. Preliminary results indicate that if n terms of the local error are estimated, then n terms of the global error expansion can be obtained.

Theoretical results for variable order variable stepsize methods were also obtained for the first time. It appears that for Adams methods applied to the scalar equation $y' = f(t,y)$, the global error e_n will satisfy

$$|e_n| \leq K\varepsilon + o(\varepsilon),$$

$$e_n = \varepsilon e(t_n) + o(\varepsilon),$$

where

$$e'(t) = f_y(t, y(t))e(t) + 1,$$

if (1) the first term in the local truncation error is positive, (2) the derivative in this term is evaluated exactly, and (3) h_{n-1} is chosen so that this term is equal to $h_{n-1}\epsilon$. The second condition makes this result impractical. However, it can be shown that this result remains true if the derivative is estimated using Milnes device.

$$\underline{y}' = \begin{bmatrix} 0 & 1 \\ -2 & 2 \end{bmatrix} \underline{y} \quad ; \quad \underline{y}(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Figure 1.

3.2 Graphic Support Packages

3.2.1 Call OS Plot Programs (A. Whaley)

Calling sequences:

PLOT (X, Y, N, SC)

REAL X(N), Y(N), SC(4)

The PLOT subroutine graphs a continuous line consisting of N-1 line segments. The N points on the line are specified by X(I), Y(I) where $1 \leq I \leq N$. Axes are put below and to the left of the plot to indicate the range of values in the data. SC tells PLOT how big the screen is in terms of X and Y values. Only one error message per call to PLOT will be given if some X, Y pairs fall outside of the window specified by SC.

Calls LINE, SEND, AXIS

SC(1) = X min

2   max

3   Y min

4   max

SCALE (X, Y, N, SC)

The SCALE routine sets SC to values so that a graph of the X, Y points exactly fills the screen. An error message is printed if all the X or all the Y values are equal. Parameters are identical to those given to PLOT.

AXIS (1 or 2, SC, IA, N)

IF = FIRST PARAM = 1 FOR HORIZ AXIS

2 VERT

SC: as provided by SCALE routine.

IA: screen coords of endpoints of axis.

IA (1) X1

(2) Y1

(3) X2

(4) Y2

N = number of tic marks with numeric values to be placed along axis.

Operation is largely self-explanatory. The values needed to label the axes are interpolated from values in SC.

Calls NCVT, SEND

If IE is negative, axis and tic marks are not drawn.

NCVT (A1, T, TL, TC)

REAL*4 A1

INTEGER TL, TC, T*2(TL)

NCVT converts a single precision floating point number (A1) into a character string which it places at the beginning of the array T. Also passed is the array dimension, TL. Returned in TC is the character count of characters placed in T. If TC is odd, the first character not included in the count is guaranteed to be a blank. Maximum length of a converted number is nine characters, but many are less.

Examples:

$\emptyset$
2
-2
2.5E1
9E25
1.7E - 6
1.012
1.012E - 36
-9.55E + 72

} worst cases

Calls CHR

CHR (M, T, TL, TC)

INTEGER TL, TC, T*2(TL)

INTEGER M/5/ or M/'bbbX'/'

CHR places the character in M into the buffer T. The variable TL gives the size of the text buffer. TC is a character count; on entering the first time it should have been set to zero. It also serves internally as a position pointer. When a character is placed in the left half of a halfword, the right half is set to a blank.

PSCALE (R, TH, N, PSC)

REAL*4 R(N), TH(N), PSC

Routine compatible with PLOT (polar plot), using same calling parameters. The maximum absolute value of the radius values (R) is placed in PSC. As zero is at the center of the display, PSC contains value of the largest circle that will fit on the screen. If PSC is .02, PLOT will put a value of .01 in R halfway along the radius (halfway to the outermost circle). Theta is in radians.

PLOT (R, TH, N, PSC)

Parameters are as described in PSCALE. PSC must be correctly set for this routine to work correctly. $R(I)$ and $TH(I)$ $1 \leq I \leq N$ form a set of points connected by straight lines. Negative values of R appear π radians away from the value in $TH(I)$, rather than being illegal.

Calls LINE, SEND, AXIS

CLOT (Z, N, M)   Contour plot.

INTEGER*2 Z(N, M)

Unlike the other plot routines, this one requires values of Z to be scaled into the range 0 to 512. One reason is to use less space than fullword real numbers. The values of Z represent the height of a sheet which is N units by M units. The sheet is displayed as a grid of lines, but is not visible from the bottom or if located behind another portion of the sheet. No numeric values are displayed. The sheet is displayed in an isometric projection. The outer boundaries of the plot form a cube which has screen coordinates as follows:

The values at the corners of the cube in terms of input data are:

Frills such as showing the underside of surfaces are deleted due to speed consideration. Line segments forming the surface grid are checked to see if they are hidden using only the value at one end. This results in some peculiar side effects, but still gives a good visual idea of the shape of the function.

Calls LINE, SEND.

PLOT Logic

The window is as follows:

Data points are mapped into the window:

$$IX = \frac{X - XL}{XH - XL} 900 + 120$$

$$IY = \frac{Y - YL}{YH - YL} 739 + 60$$

Movement to the first data point is made with ISW=0 to avoid drawing a line.

Subsequent calls are made with ISW=1, to connect the data points with straight lines on the screen.

```

1.000 SUBROUTINE PLOT (N,YN,SC)
2.000 DIMENSION X(N),Y(N)
3.000 REAL*4 SC(4)
4.000 INTEGER IA1(4)/120,60,1019,60/,IA2(4)/120,60,120,799/
5.000 XL=SC(1)
6.000 XH=SC(2)
7.000 YL=SC(3)
8.000 YH=SC(4)
9.000 YDIF=YH-YL
10.000 XDIF=XH-XL
11.000 IF (XDIF.EQ.0.0) GO TO 50
12.000 IF (YDIF.NE.0.0) GO TO 20
13.000 50 WRITE (6,30) XDIF,YDIF
14.000 30 FORMAT (' PLOT: IMPOSSIBLE: XSPAN=',E14.6,', YSPAN=',E14.6)
15.000 RETURN
16.000 20 XR=900./XDIF
17.000 YR=240/YDIF
18.000 ISW=0 739.
19.000 IERR=0
20.000 DO 40 I=1,N
21.000 XI=X(I)
22.000 YI=Y(I)
23.000 IF (XI.LT.XL) GO TO 90
24.000 IF (XI.GT.XH) GO TO 90
25.000 IF (YI.LT.YL) GO TO 90
26.000 IF (YI.GT.YH) GO TO 90
27.000 IX=(XI-XL)*XR+120.
28.000 IY=(YI-YL)*YR+60.
29.000 CALL LINE (IX,IY,ISW)
30.000 ISW=1
31.000 GO TO 40
32.000 90 IF (IERR.EQ.1) GO TO 40
33.000 IERR=1
34.000 WRITE (6,92)
35.000 92 FORMAT (' PLOT: SOME POINTS OUT OF WINDOW')
36.000 40 CONTINUE
37.000 CALL AXIS (1,SC,IA1,5)
38.000 CALL AXIS (2,SC,IA2,5)
39.000 CALL SEND
40.000 RETURN
41.000 END
0 CO.AXIS

```

```

L
1.000 SUBROUTINE SCALE (X,Y,N,SC)
2.000 DIMENSION X(N),Y(N)
2.500 REAL*4 SC(4)
3.000 REAL B/1E65/
4.000 XL=B
5.000 XH=-B
6.000 YL=B
7.000 YH=-B
8.000 DO 10 I=1,N
9.000 IF (X(I).LT.XL) XL=X(I)
10.000 IF (X(I).GT.XH) XH=X(I)
11.000 IF (Y(I).LT.YL) YL=Y(I)
12.000 IF (Y(I).GT.YH) YH=Y(I)
13.000 YDIF=YH-YL
14.000 XDIF=XH-XL
15.000 IF (XDIF.EQ.0.0) GO TO 50
16.000 IF (YDIF.NE.0.0) GO TO 20
17.000 50 WRITE (6,30) XDIF,YDIF
18.000 30 FORMAT (' SCALE IMPOSSIBLE: XSPAN=',E14.6,', YSPAN=',E14.6)
19.000 RETURN
20.000 20 SC(1)=XL
21.000 SC(2)=XH
22.000 SC(3)=YL
23.000 SC(4)=YH
24.000 RETURN
25.000 END

```


AXIS Logic

First the axis line is drawn between the endpoints specified in IA. The axis line and tic marks are drawn only if the first parameter (IE=±(1 or 2)) is positive. ID is set to IABS(IE) and is 1 for horiz axis, 2 for vert axis. ID is used to tell which coordinate to change in order to make a tic mark (10 units), and to select a set of values from ISUB which tell how much to move from the axis/tic intersection to be at a convenient place for the numeric labels.

$$\text{ISUB}(1, \text{ID}) = \Delta X$$

$$\text{ISUB}(2, \text{ID}) = \Delta Y$$

IB is the distance to the end of the axis line:

IS is the spacing between tic marks. AB (A-Begin) is the value of user data at the beginning axis. AS is the amount to be added to AB in order to arrive at each successive tic mark. $AB + nAS$ is the value displayed on the graph.

$$0 \leq n < N.$$

```

1.000 SUBROUTINE AXIS (IE,SC,IA,N)
2.000 REAL*4 SC(4)
3.000 INTEGER IA(4)
4.000 INTEGER T(2), T2(2), T3(2)
5.000 INTEGER TXT*2(12), ISUB(2,2)/72,34,108,6/
6.000 ID=IABS(IE)
7.000 IF (IE.LE.0) GO TO 23
8.000 CALL LINE (IA(1),IA(2),0)
9.000 CALL LINE (IA(3),IA(4),1)
10.000 23 CONTINUE
11.000 IB=IA(ID)
12.000 AB=SC(ID+ID-1)
13.000 AS=(SC(ID+ID)-AB)/N
14.000 IS=(IA(2+ID)-IB)/N
15.000 T(3-ID)=IA(3-ID)
16.000 T2(3-ID)=T(3-ID)-10
17.000 NP1=N+1
18.000 T3(3-ID)=T(3-ID)-ISUB(3-ID,ID)
19.000 DO 19 I=1,NP1
20.000 T(ID)=IB
21.000 T2(ID)=IB
22.000 T3(ID)=IB-ISUB(ID,ID)
23.000 IB=IB+IS
24.000 CALL NCUT (AB,TXT,12,IN)
25.000 CALL TEXT (T3(1),T3(2),TXT,6,IN)
26.000 AB=AB+AS
27.000 IF (IE.LE.0) GO TO 19
28.000 CALL LINE (T(1),T(2),0)
29.000 CALL LINE (T2(1),T2(2),1)
30.000 19 CONTINUE
31.000 CALL SEND
32.000 RETURN
33.000 END

```

NCVT Logic

The number A1 in 360 internal machine code is used to derive a string of readable characters. Exponential form is used unless the exponent is zero.

The first task is to multiply or divide A1 by powers of 10 until A1 is in the range of $1.00000 \leq A1 < 10.0000$. At the same time, IEXPO is kept; if A1 (brought into range) is multiplied by $10.**IEXPO$, the original A1 is formed. Thus IEXPO is the exponent to be printed. In order to save time, and some accuracy, Table 1 is kept in the arrays GE, LT, EXPN, and FACTOR. FACTOR(I) is a number to be divided into A1 (power of 10). EXPN(I) is the correction to IEXPO for that operation. Note that more than one operation will frequently be required to bring an arbitrary number in range.

INDEX	LT	GE	EXPN	FACTOR
1	7	3	0	1
2	0	-2	+1	10
3	2	4	+2	100
4	-3	5	+4	10000
5	-4	-5	+8	10^8
6	-7	-6	-1	.1
7	8	6	-2	.01
8	9	-7	-4	.00001
9	-9	-8	-8	10^{-8}

Table 1.

Control is provided by the arrays LT and GE. These arrays use FACTOR to make comparisons on A1 and determine what division operations

should be performed. They also indicate when the number A1 is in range (LT(Z)=0).

Sequence of control is as follows. An initial index IR=1 is selected. Then an initial Hunt is begun. During the hunt a single appropriate scaling factor is sought. After one is found, another hunt is begun until a final hunt determines the number to be in range. (A) With the hunt started at IP-1, (D) FACTOR(IP) is checked to see if A1 is less than (LT) or greater than (GE) or equal to FACTOR. Then the array LT or GE is checked after its value is saved in IN (and absolute value saved in IP) to determine what action to take. If IN is now zero, the number A1 is in range and we quit. If IN is positive we proceed to (B) to form the next check on A1 with a new value from FACTOR. If IN is negative, IP is an index into a value of FACTOR that should be divided into A1. After IEXPO is updated, we proceed to (A) for the next hunt. All possible paths are outlined in Diagram 1.

Forming the number into characters is performed as follows. If the number A1 is negative, a minus sign is output via CHR. Then the absolute value A is put through the above procedure to form the magnitude A and the exponent IEXPO. (C) Then the integer portion of the mantissa is output as a character (via CHR again). If this is the second numeric digit, a decimal point is output first. Then the remaining part of the mantissa is multiplied by ten. If this part is zero, or five characters have already been output, we quit. Otherwise, return to (C).

Next the exponent is printed, be it not zero. First an E is output, then a minus sign if needed. The tens digit of the exponent is output if not zero, then the units digit is output.


```

34.000 SUBROUTINE NCUT (A1,T,TL,TC)
35.000 INTEGER EXPN(9)/0,1,2,4,8,-1,-2,-4,-8/
36.000 INTEGER LT(9)/7,0,2,-3,-4,-7,8,9,-9/
37.000 INTEGER GE(9)/3,-2,4,5,-5,-6,6,-7,-8/
38.000 REAL*8 FACTOR(9)/1.,10.,100.,10000.,100000000.,1.,.01, %
39.000 .001, .00000001/
40.000 REAL*8 A .0001
41.000 INTEGER TL,TC,T*2(TL),M,' '
42.000 INTEGER DOT,' ',E,' '
43.000 A=ABS(A1)
44.000 TC=0
45.000 IF (A1.LT.0) CALL CHR (M,T,TL,TC)
46.000 NDOT=TC+1
47.000 IEXPO=0
48.000 IF (A.EQ.0) GO TO 40
49.000 60 CONTINUE
50.000 IP=1
51.000 50 CONTINUE
52.000 IF (A.LT.FACTOR(IP)) GO TO 20
53.000 IN=GE(IP)
54.000 GO TO 30
55.000 20 IN=LT(IP)
56.000 30 IP=IABS(IN)
57.000 IF (IN.EQ.0) GO TO 40
58.000 IF (IN.GT.0) GO TO 50
59.000 A=A/FACTOR(IP)
60.000 IEXPO=IEXPO+EXPX(IP)
61.000 GO TO 60
62.000 40 IA=A
63.000 IF (TC.EQ.NDOT) CALL CHR (DOT,T,TL,TC)
64.000 CALL CHR (IA,T,TL,TC)
65.000 IF (TC.GE.5) GO TO 95
66.000 A=(A-IA)*10.
67.000 IF (A.NE.0) GO TO 40
68.000 95 IF (IEXPO.EQ.0) RETURN
69.000 CALL CHR (E,T,TL,TC)
70.000 IF (IEXPO.LT.0) CALL CHR (M,T,TL,TC)
71.000 IEXPO=IABS(IEXPO)
72.000 IA=IEXPO/10
73.000 IF (IA.NE.0) CALL CHR (IA,T,TL,TC)
74.000 IA=IEXPO-IA*10
75.000 CALL CHR (IA,T,TL,TC)
76.000 RETURN
77.000 END

```

CHR Logic

The character count TC is converted into halfword index:

$$TC1 = TC/2 + 1$$

A remainder in the division indicates the right half of the halfword is to be used; the blank already there is subtracted out and the new character added in.

If the character is an integer 0-9, it is converted into the character 0-9 by adding 240.

```

78.000 SUBROU THE CHR (M,T,TL,TC)
79.000 INTDCL M,TL,TC,T*2(TL),TC1
80.000 M1=M-(M/256)*256
81.000 IF (M.NE.M1) GO TO 10
82.000 M1=M1+240
83.000 10 CONTINUE
84.000 TC1=TC/2+1
85.000 TC=TC+1
86.000 IF (TC.NE.TC1*2) GO TO 20
87.000 T(TC1)=T(TC1)+M1-64
88.000 GO TO 30
89.000 20 T(TC1)=M1*256+64
90.000 30 RETURN
91.000 END

```

CPLOT Logic

The location of a point of the sheet on the screen is formed as follows:

A Horizon is kept to indicate the height the sheet has attained so far in screen coordinates:

As each point of the sheet is calculated, it is referred to as a Z checkpoint. All points on the horizon are filled in with this Z value where it is greater than the horizon. Only the Z value is checked even though it is an entire line that may be partly or completely obscured by the rest of the sheet. Two lines are drawn to each Z point if it is above the horizon, one in the X direction, one in the Y direction:

The lines are of course not at right angles, but are drawn between the screen coordinates of the current and previous Z checkpoints (previous in two directions).

GRASS SYSTEM: OPERATION CRITICAL
REMOTE COMMUNICATION

18: 29: 36

```

100.000 SUBROUTINE CPlot (Z,N,M)
101.000 INTEGER*2 Z(N,M), ZMAX(762)
102.000 X=512./N
103.000 Y=250./M
104.000 DO 10 I=1,762
105.000 10 ZMAX(I)=0
106.000 DO 20 I=1,M
106.100 IYP=IY
106.200 IY=I*Y
106.300 IZM=IY-1
107.000 DO 30 J=1,N
109.000 IX=IFIX(J*X)
110.000 IZ=Z(J,I)
111.000 IXPOS=IY+IX
112.000 IZPOS=IZ+IY
113.000 IF (IZPOS.LT.ZMAX(IXPOS)) GO TO 60
113.100 70 IZM=IZM+1
113.200 ZMAX(IZM)=IZPOS,Max 0((Zmax(IZM)), IZPOS)
113.300 IF (IZM.LT.IXPOS) GO TO 70
114.100 IF (J.LE.1) GO TO 40
115.000 CALL LINE (IOXPOS,IOZPOS,0)
116.000 CALL LINE (IXPOS,IZPOS,1)
116.050 40 CONTINUE
116.100 IF (I.LE.1) GO TO 50
118.000 IZP=Z(J,I-1)
119.000 IOXPP=IYP+IX
120.000 IOZPP=IZP+IYP
121.000 CALL LINE (IOXPP,IOZPP,0)
122.000 CALL LINE (IXPOS,IZPOS,1)
122.100 GO TO 50
122.200 60 IZM=IXPOS

```

RETURN

GRASS SYSTEM: OPERATION CRITICAL
REMOTE COMMUNICATION

18: 29: 47

123.000 50 CONTINUE
124.000 IOXPOS=IXPOS
125.000 IOZPOS=IZPOS
126.000 30 CONTINUE
127.000 20 CONTINUE
128.000 CALL SEND
129.000 RETURN
130.000 END

-71-

RETURN

ITS

STOP
44.12 SEC 1.94 SERVICE UNITS

3.3 User Oriented Software

3.3.1 Global Macro Analysis (T. Runge)

The Global Macro Analysis system, GLOBAL, has been coded in PLW and is presently being debugged. This system takes a user's definition of a network in terms of the macros involved and produces the output routines S1, S2, and DIFFUN for the numerical simulation of the network. Global Macro Analysis consists of three processes:

- 1) The input routine. This reads the cards and interprets the information to create PLW data structures representing the network.
- 2) A pre-end order traversal of the structure of macros which is the user's network. This traversal produces an intermediate (tree) representation of the network's equations and gathers information to be used to print a network variable map upon request.
- 3) The output routine builder. This processes the intermediate representation of the equations and determines the output classes to which the variables belong. It then constructs S1, S2, and DIFFUN from the network equations.

Input Routine

This routine was coded prior to the present report period and has been described in some detail in the previous two Quarterly Progress Reports. During this report period a new feature has been added and the input routine has been debugged.

The new feature allows the user to declare functions and describe their syntax so that he may use these functions (in addition to the built-in functions) in the equations of his network.

These function declarations must appear immediately after the terminal type definitions in the input stream. At present they must appear one per card and adhere to a fixed format. As will be mentioned below, this

fixed format will be relaxed in the future.

Each function which the user declares he must also make available to the simulation package in the form of a Fortran function subroutine so that the function occurrences can be evaluated.

Network Traversal

This process also was coded prior to the present report period and has been described previously. During this report period the preorder visit routine, VISITD, has been debugged. The network traversal will be finished when the debugging of the endorder visit routine, VISITU, which is presently underway, is complete.

Output Routine Builder

This routine also was coded prior to the present report period and has been described previously. It is scheduled for debugging as soon as the network traversal is finished.

At present, a number of improvements to the Global Macro Analysis system are under consideration. They include:

- 1) Freeing the fixed format presently required for user declaration of functions.
- 2) Consolidation of data from the declarations block and the parameter equations block into a single PDP-8 block which would contain all declarations of local and global variables, parameters, and default parameter assignments. This would allow the user to display all declarations for a given network element (macro) on the screen at one time.
- 3) Allowing the use of arrays in the network equations.

These new features will be implemented when the Global Macro Analysis system, as it now stands, is debugged.

During this quarter, research was begun on the development of a more general macro analysis system which would be capable of producing output for simulation by a number of presently available packages - for instance, DYNAMO, CSSL, GPSS.

This system, as presently envisioned, would consist of three parts, roughly corresponding to the three processes in terms of which we have described Global Macro Analysis. These are:

- 1) A general data structure in which could be modeled all of the kinds of systems which we wish to simulate.
- 2) A synthesis program which would expand the macro instances into a tree structure of the network elements.
- 3) A translator driven by rules specific to the intended simulation package.

This generates string input for execution by the simulation package.

Data Structure

The general data structure desired seems in many ways similar to the data structure used by the present Global Macro Analysis system. Its features are described below.

1. Terminal Type Table

This table defines, for each terminal type used in the network, the names and types of the terminal variables defined on a terminal of that type. The present system also uses a terminal type table.

2. Function Declaration Table

This table declares the names and defines the syntax of functions which the user will be supplying to the simulation package. Declarations and syntax definitions for functions which are predefined in the intended simulation package will be available to the system from its own library. The present system also uses a function declaration table.

3. Macro Definitions

The major components of a macro definition are:

- a) (Subpicture Block in present system)

Names of macros invoked by this macro and non-default parameter assignments to be used for each invocation.

- b) (Terminal-Connections Block in present system)

Descriptions of the terminals defined in this macro and of those brought up from invoked macros. Description of the connections between these terminals.

- c) (Declarations Block and Parameter-Equations Block in present system)

Declarations of global and local variables and default parameter assignments for this macro. Macro text, which is a generalization of the macro (arithmetic) equations of the present system.

The macro text might be arithmetic equations, logical equations, a program segment in some language, or other possibilities which will be investigated.

4. Terminal and Terminal Variable Referencing

Two new features will be provided to make the referencing of terminals and of terminal variables easier for the user.

- a) The user can attach names to terminals for the purpose of implicitly specifying a connection between two terminals (within the same macro) to which the same name is attached.

- b) The user can attach names to terminals for the purpose of referencing terminal variables by means of these names. The user must also specify which terminal variable he wishes the name to reference. Thus attaching $V=E$ to terminal 0 would

allow the user to refer to V rather than to E(0) in the macro equations.

In both of these cases, the attached names would be placed next to the terminal in the screen display and included in the Terminal-Connections Block portion of the data structure.

5. Strip Terminal

A new kind of terminal, the strip terminal, will be provided. A strip terminal will act like a variable number of ordinary terminals in that any number of ordinary terminals (of the same type) can be connected to it and it will cause the generation of one ordinary terminal to receive each connection.

This allows the user to construct a macro with a variable number of external terminal connections without his having to short all of these terminals together.

If terminal 0 is a strip terminal of a type with a terminal variable named E on it, the user can refer to the variable E on the i^{th} terminal generated from the strip terminal by E(0,i). The new system will also provide operators for symmetric handling of terminal variables on a strip terminal. Examples of such operators which could be used in the network equations are:

$$\text{STMIN}(E(0)) = \text{MIN}(E(0,1), E(0,2), \dots, E(0,n))$$

$$\text{STMAX}(E(0)) = \text{MAX}(E(0,1), E(0,2), \dots, E(0,n))$$

$$\text{STSUM}(E(0)) = E(0,1) + E(0,2) + \dots + E(0,n)$$

$$\text{STPROD}(E(0)) = E(0,1) * E(0,2) * \dots * E(0,n)$$

where terminal 0 is a strip terminal with n connections to it and with a terminal variable E defined on each.

Names may be attached to strip terminals for specifying implicit

connections. Only a strip terminal may have more than one such name attached to it.

There is a restriction placed on the use of strip terminals. Strip terminals in any macro may only be used externally. That is they may only be connected to terminals in the macro invoking that macro. This restriction is consistent with the intention in defining the strip terminal to allow one macro to receive a variable number of connections from terminals defined in another macro. When defining a macro, a user should know precisely what terminal connections he wants between terminals within that macro and have no use for a local strip terminal.

It is worth noting that a strip terminal itself does not define any terminals. It is the connections to the strip terminal which generate terminals. Thus a connection between two strip terminals is meaningless and is not allowed under the restriction above.

6. Terminal Variable Types

Some research has been done into the usefulness of allowing the user to define his own terminal variable types in terms of how they act when terminals are connected. As of this writing, we have not found any types of terminal variables other than the familiar E and I types which seem to be generally useful. Unless and until such terminal variable types are found, the new system will restrict itself to the predefined E and I types of the present system.

Synthesis

This process will produce a tree representation of the network. Each tree node represents one macro instance and includes the following:

- i) non-default parameter assignments passed down from invoking macro

- ii) default parameter assignments for those parameters which have no non-default assignment passed down
- iii) the macro text
- iv) a symbol table for the global and local variables, parameters, and terminal variables
- v) a list which gives, for each terminal ID number, the location in the symbol table of the associated terminal variables
- vi) the equations generated for the rings of connected terminals.

The synthesis process also constructs a terminal table for its own use. This table is used to reference terminal variables and to generate the ring equations.

The tree nodes are built during a pre-end order traversal of the network's tree structure.

Preorder Visit

During the preorder visit to a macro parts i, ii, and iii of the tree node are formed.

Also, the global and local variables, the parameters, and any names attached to terminal variables are entered in the symbol table.

Also, the terminal table is formed. The connections between terminals are encoded in the form of connection rings - circular linked lists of connected terminals. Connections of external terminals to internal terminals in the invoking macro are encoded by linking the external terminal's terminal table entry to the terminal table entry for the internal terminal (in the invoking macro's terminal table).

As the terminal table is formed, external strip terminals are expanded into the appropriate number of ordinary external terminals - one to receive each connection. These generated external terminals are linked to

the connected internal terminals.

Endorder Visit

During the endorder visit to a macro the remainder of the tree node is constructed.

Terminal types are passed along the lines of external connections. When any connection is specified, the types of the two terminals involved must match if both are defined. If only one is defined, that type is passed to the other terminal. When this type passing is complete, all terminals in this macro should have a type defined.

At this point the terminal variables for each terminal are entered in the symbol table, completing part iv of the tree node. As this is done, the list which is part v of the tree node is constructed.

Finally, the ring equations for the rings of connected terminals are generated and placed into the tree node.

We also form equivalence rings (of variables which are equivalent) in the symbol table. The symbol table entries for names attached to terminal variables will be linked to the symbol table entries for those terminal variables. The symbol table entries for E type terminal variables on connected terminals will also be linked as equivalent.

We note here that any variables which are pre-defined in the intended simulation package are entered in the symbol table as globals before the network traversal begins. Thus they will be defined when they are used in parameter assignments or macro text.

Translator

The translator will process the tree representation of the network formed by the synthesizer and generate the string output necessary to drive the intended simulation package. In doing this it may call upon certain

procedures from a library of translation procedures for the various simulation packages.

Much research is still needed to fully develop the translator.

However, we can already recognize a number of operations which it must be able to perform.

Syntax Analysis

The translator will check the syntax of the parameter assignments and of the macro text. This will be done with the aid of the function syntax table and either a syntax definition or a syntax analyzer for the intended simulation package. During the syntax analysis all variable occurrences will be replaced by pointers into the symbol table.

Types of Simulation Packages

We will want to support two vastly different types of simulation packages - Problem Oriented Packages (i.e., present system) and Procedural Packages (i.e., DYNAMO). The operations which the translator must perform for these two cases have great differences. Some of these operations are described below.

Problem Oriented Packages

For such a package the macro text constitutes a set of simultaneous equations which will be solved by the simulation package. These equations must in general be converted to assignment statement form and reordered so that all variables are assigned on a left hand side before being used on a right hand side. This will require invoking some or all of the operations below.

- i) Variable elimination. When an equation of the form $A=B$ is found, A and B are placed on the equivalence ring in the symbol table and the equation is discarded.

- ii) Variable classification by computational dependency. The G and T type variables of the present system are examples of this. Certain variables are initially known to be of these types. Then any variable which is found in some equation to depend only on G type variables and constants becomes a G type variable. Any variable found in some equation to depend only on G and T type variables and constants becomes a T type variable.

When a new variable is reclassified, the equation which led to the reclassification is converted to assignment statement form with that variable isolated on the left side. The new assignment statement is then placed at the end of a list of assignment statements - insuring that all variables on its right hand side have values assigned to them in a previous assignment statement.

- iii) Variable classification by operators used on the variable. The Y type variables of the present system are an example of this. Any YL variable which is differentiated, is a function argument, is operated on by exponentiation, or has a coefficient of type YL or Y, becomes a Y type variable. This classification allows these variables to be treated in special ways by the simulation package.
- iv) Generation of new left hand sides. Equations of the form $LHS=AHS$ can be converted to the form $Di=RHS-LHS$ with a new variable Di introduced. Then the simulation package will iteratively drive all of the Di 's to 0.

Procedural Packages

For procedural packages the macro text is program code in some language. Thus any equations will be in the proper assignment statement form in the macro text. The translator will be able to reorder the assignment statements so that variables are assigned before they are used on a right side if necessary.

Some operations which will be available for simulation packages of either type are:

- i) Partitioning of text into disjoint sets to be handled differently by the simulation package.
- ii) Conversion of text into a set of fixed formats. Such formatting is necessary for assignment statements in DYNAMO.
- iii) A name generation process for variables. Names can be generated in different formats for different types of variables. Only one name is generated for any equivalence ring in the symbol table.

3.3.2 The PL-F Compiler (L. Lopez)

The PLW compiler which has been in use for some time has several shortcomings such as lack of recursion, dynamically dimensioned arrays and almost a total lack of facilities to allow the modification of large programs. Specifically we are at the level of FORTRAN in this respect. Hence PL-F, the second version, will be described. This document will define the methods used to provide the above and will serve as a definition of the major components of the run time representation. Such a definition will provide a communications media for other interested parties and will help also to freeze the design of

this compiler. Much time has been lost due to late arrival of this document.

Efficient Modifications

The usefulness of functional modularity is well known. Less often noted is the desirability of the efficient modifiability which often results from this. A small change to a large program should not necessitate the recompilation of the entire program. FORTRAN and to a less extent PL/I provide this feature. Burroughs extended Algol on the 5500 did not. The necessity of this feature is dependent greatly on the size of the program in question, the speed of the compilation process, and the frequency of change necessary. Since this compiler is intended for in house debugging of systems of programs it is expected that the frequency of changes as well as the overall size of programs will be large. The speed of the present PLW compiler is about 4000 cards per minute. It is not expected that PLF will run as fast as this. Large amounts of computer time are used for compilation on the Burroughs machine due to the inflexible nature of the modularity. A change to one procedure necessitates a complete recompilation. In the newer machines (6700) a separately compiled procedure may be bound into other programs. This in essence is coming around to my point of view.

PL/I

Although PL/I allows recompilation of external procedures it does not allow recompilation of internal ones. This is in effect saying that structured programming and nesting is good and useful unless modification is necessary.

What we are attempting to implement is the modification of internal procedures in the original environment.

The FORTRAN Environment

The target language for PLW and PLF is a subset of FORTRAN compatible

with the Burroughs 6700, DEC PDP 10 and the IBM 360. These three machines, although they were chosen more for their availability rather than for any other characteristics, do seem to have a sufficiently typical FORTRAN implementation so as to achieve the goal of portability. A specific definition of the subset has not been made as it is very infrequent that there is any question of compatibility problems. This is chiefly because FORTRAN is a very standardized language. It is this standardization which should give up portability.

Numerical problems require the compiler to emit FORTRAN code which is dependent on the target machine desired. This problem is understood more to be a change in the mapping of PLF declarations into FORTRAN declarations rather than an entirely new aspect. Hence implementation of this feature will be postponed until appropriate.

Static Common

PLF will use FORTRAN named common to contain constants, lists, static variables as well as static arrays. This area will also contain what are loosely called system variables (SV). These locations provide for the communication of various information throughout the object programs. Certain SV's are at fixed displacements in to the SC so as to allow easy accessibility. This space taken as a whole will be called Static Common (SC). The size of SC is known at compile time as opposed to dynamic common whose size is determined at run time.

Dynamic Common

Dynamic Common (DC) will be allocated at run time by means appropriate to the target machine. The limits of the DC are given by two system variables called BOTTOM and LAST. From LAST to the system variable NEXT is known as the allocated user area (UA). In the UA are located any storage used by the user

for the creation of DSECT's. (DSECT's are storage layouts for the implementation of list processing algorithms). From BOTTOM to TOP is an area known as the run time stack (RTS). In this area are allocated dynamic arrays (DA) and activation records (AR) for recursive procedures.

From TOP to NEXT is free space. NEXT can expand down and TOP can expand up. The words at TOP and NEXT are also free.

The indices discussed above, and in fact almost all indices to variables, lists, etc. are word displacement off of the SC. The mechanism used to obtain the DC also provides these indices.

DC Allocate

The dynamic common allocate routine (DCA) obtains the DC and initializes TOP, BOTTOM, NEXT and LAST as shown in Figure 2.

Figure 2.

On the IBM 360 the DCA will obtain most of the free storage available in the REGION by means of a V type GETMAIN SVC. It will then insert into the SV's the appropriate index values.

On the DEC PDP 10 and the Burroughs B6700 the DCA will not be employed; instead these machines will have a 64K (K=1024 words) DC already set up via COMMON statement. A BLOCK DATA statement will be used to initialize

the system variables. Although such a method is very expensive on the IBM machine it is a reasonable way to proceed on these two virtual machines. The 64K limit is set by the array indexing limits on the Burroughs machine. Such a limit does not exist on the DEC 10 or IBM 360.

Procedures

Procedures are numbered from 1 to n. Within each procedure the return point from a call and the entry points for a call are numbered from 1 to m. Hence (i,j) specifies a return or call to part j of procedure i. The (i,j) is called a procedure reference word (PRW). The i is the procedure number; the j is an entry number. The j is used to index into a computed GO TO to reach the appropriate code. This computed GO TO is called the procedure reference word computed GO TO (PRWCG) and one exists for each procedure.

FORTTRAN Modules

The PLF compiler produces three types of modules; the MAIN\$ module, the PATCH\$ module and the P\$i modules. These modules together with the initialization of the SC from the initialization file provide the efficient implementation of program modification. The initialization file also is produced by the PLF compiler.

The INIT\$, MAIN\$, PATCH\$, P\$i and the SC

The INIT\$ is a subroutine which initializes static common (SC) from the initialization file and also calls on the CORE\$ procedure to obtain the dynamic common (DC).

When a PLF program is compiled for the first time the MAIN\$ subroutine as well as an empty PATCH\$ subroutine is created. The initialization file is also created at this time.

When a PLF procedure i is recompiled a P\$i subroutine is created

(or recreated), the PATCH\$ subroutine is recreated (to call on the P\$i) and the initialization file is recreated (to reflect changes due to the new procedure).

In the initialization file are, among other things, the procedure reference words (PRW). These PRW's are changed from (i,j) to (i',j'). To reflect the new placement of entry points and calls within procedure i, j is changed to j'. If i was less than 1000 then a new i > 1000 is assigned for i' to note that procedure i was changed. If i > 1000 then i had already been patched so i need not change.

The subroutine MAIN\$ has a computed GO TO called the Main Subroutine Computed GO TO (MSCG) which, given i, transfers control to the code for procedure i. If i > 1000 then the procedure has been patched and i is the procedure number of the patched procedure. The MSCG falls thru since i is too big. The statement after the MSCG is a call to PATCH\$. PATCH\$ subtracts 1000 from i and uses it to index into the PATCH Subroutine Computed GO TO (PSCG). The GO TO transfers control to a call statement which calls procedure P\$z where $z = i - 1000$. When P\$z returns to PATCH\$ it is either returning or initiating a call. In both cases PATCH\$ returns to MAIN\$. MAIN\$ then branches to the code which extracts i and performs the (MSCG).

Parameter Lists

The parameter list generated for PLF procedures contains a PRW to the called and calling procedures in the following format:

WORD	CONTENTS	USE
1	(i,j)	i = procedure to call j = entry point
2	(i*,j*)	i* = procedure to return to j* = return point of procedure


```

3      parameter'
      .
      .
      .
l      last parameter
l+1    0

```

In parameterless procedures words 3 to l+1 will be left out.

In order to return from a procedure a call with (i*,j*) as word 1 of a parameter list should be used. The pointer to the parameter list is a SV called PARM. Thus to perform a return PARM would be set to the value of PARM at procedure entry +1. Note that this return does not delete storage from the run time stack or anything other than transfer control back to the caller. Note that PARM is not preserved around a call statement.

Activation Records

Recursive procedures will be implemented using the activation record (AR) technique. In this method the fixed length AR is obtained upon procedure entry. The steps followed for a call are:

- 1) The PSCG or MSCG causes the execution of FORTRAN representation of the PLF procedure.
- 2) The procedure extracts j from the parameter lists and uses the PRWCG to get to the correct entry point.
- 3) The space for the AR is obtained. The procedure saves some system variables such as TOP, PARM, etc. in the AR.
- 4) The code necessary to initialize any constants in the AR is executed.

The storage for fixed sized dynamic arrays and simple variables are allocated in the AR at compile time. Hence,

- 5) The addresses (which are actually indices into common) of parameter lists to fixed sized dynamic arrays in this AR, simple variables in this AR, all parameters to this call and all other variables to other procedures which

contain this procedure are initialized.

- 6) The expressions, etc. which define the lengths of arrays, etc. are executed.

This code may cause procedure calls. One may not reference any varying length arrays which we are attempting to allocate in any such code since such arrays are non-existent. Code such as:

```
P:  PROC    RECURSIVE;

      DCL    Z(I);

      Q:  PROC RETURNS (INTEGER);

            Z(1) = 100;

            RETURN    Z(1)

      END Q;

      DCL    Y(Q);

      END;
```

is invalid. Note that a compiler check for this is not done as it would seriously restrict this type of code. If I was a constant extra code to be executed.

- 7) All references to the arrays allocated in 6) are generated.

It is noted that all references to variables located in DC cause some

Non Recursive Procedures

It is evident that a great deal of time can be taken up at procedure entry. While such time is unfortunately necessary for recursive procedures, it is not for non recursive procedures. All simple variables for non recursive procedures are static (i.e. in SC). The code to reference a dynamic simple variable once takes up more space than the variable, hence no space would have been saved. All static arrays are placed in SC. The pointers, etc. to fixed length dynamic and varying length dynamic declared in a non recursive procedure

are in the SC. All parameter lists to other procedures used by a non recursive procedure are in the SC.

The non recursive procedure (and the recursive procedure) handle for the most part, the prologue code necessary to exit the procedure. Hence an AR is not necessary for this reason. In order to provide trace backs an AR in static common is provided by the compiler for non recursive procedures. The SV pointer to present activation record (PPAR) is used to locate the AR of the presently executing procedure. It is also noted that for every procedure a pointer to its most current AR exists (PMCAR).

α

AR Format:

WORD	USE
1	PPAR on entry
2	PARM on entry
3	TOP on entry
4	PMCAR on entry
5	Other saved SV's needed
⋮	
K	PARM 1
⋮	
K+ℓ-1	Last PARM
K+ℓ	Var 1, etc.
⋮	⋮
END of AR	

This value is saved in the AR and the PMCAR set to the pointer to this AR. The purpose of this value is to allow internal procedures to address the current global variables.

Initialization

The initialization of the parameter lists, etc. could be done by straight code in FORTRAN. This would tend to be expensive in space. A faster and shorter method is to use lists which describe the initialization necessary and then pass this list to an assembly language program. (For simplicity in getting this system up on other machines we require this routine to be capable of being written in FORTRAN.)

An interesting observation can be made. Let P and Q be procedures and P be internal to Q. Let P reference one of Q's variables Z in a parameter list, R. If initialization of Z in R was done by straight code, then this code must not be in Q, otherwise recompilation of P might cause Q to be initializing a non-existent R. If initialization of Z in R is done by list, then we can change Q's list to reflect this change.

Constant Parameter Lists

A parameter list which references only static variables and arrays whose pointers are in SC is constant. It should be noted that the representation of an array in a parameter list is the index to the index to that array in common. The representation of a simple variable is an index to the simple variable.

All simple variables are referenced by value result or value except for character strings which are treated like arrays.

In the AR an array is represented by its index.

3.3.3 Network Analysis Package Modifications (C. Collins)

In order to achieve a greater degree of machine independence in the 360 end of the Network Analysis Package, a number of modifications to machine dependent components in the system has been undertaken. COMPIL is a routine

originally written in 360 assembler language. Its function was the generation of up to four machine language executable subroutines, doing various operations on three vectors. These subroutines were used in the solution of a sparse matrix system. COMPIL has been rewritten in FORTRAN, and now generates a vector containing up to four interpretable subroutines in pseudo code. An interpreter for the pseudo code has also been written in FORTRAN.

From the user viewpoint

The calling sequence to COMPIL and the nature of the operations performed has changed from the original. The operations are now one, two, or three address assignment statement type. So the calling sequence is of the form:

```
CALL COMPIL (OPCODE, SUBR#, DEST, LEFT, RIGHT, PW, DY, DOUT)
```

PW, DY and DOUT are the three vectors to be operated on, PW being REAL*4, the other two being REAL*8. In the calling sequence, however, the DOUT parameter must be a scalar indicating the address to be used in place of the DOUT vector in the immediate execution of the operation. DEST, LEFT and RIGHT are integers indicating the indices for the three operands, the destination, the left hand operand and the right hand operand, i.e.:

```
VEC1(DEST) = VEC2(LEFT) <op> VEC3(RIGHT)
```

SUBR# is the number of the subroutine being generated, between 1 and 4. OPCODE is an integer indicating which of the operations is to be compiled. The current op codes are tailored to the needs of the various routines calling COMPIL, and it is expected that any new operations being introduced into the system would require additions to COMPIL. TØ is a temporary REAL*8 location the different ops might use. The current op codes, 18 in number, are:

1	DOUT	←	DY/PW
2	TØ	←	DY*(-PW)
3	TØ	←	DOUT*(-PW)
4	DY	←	DY+TØ
5	DOUT	←	Ø
6	DOUT	←	DY+TØ
7	DOUT	←	DOUT+TØ
8	DOUT	←	-DOUT+TØ
9	DOUT	←	DOUT-DY
10	DOUT	←	DY-TØ
11	DOUT	←	DOUT+DY
12	DOUT	←	DY
13	PW	←	PW/PW
14	TØ	←	PW*(-PW)
15	PW	←	PW+TØ
16	TØ	←	PW*(-TØ)
17	TØ	←	DOUT
18	PW	←	PW+PW

The four subroutines generated are referenced by the entry points MATIN1, MATIN2, MATIN3 and MATMUL.

There also are five special op codes that generate no pseudo code. These special codes are indicated by calling COMPIL with SUBR#=-1. These codes have the following meanings:

Ø = Terminate all subroutines.

1 = Initialize COMPIL.

2 = Mark Pivot division.

3 = Mark forward elimination.

4 = Mark back substitution.

Op codes 2, 3 and 4 are for use in error recovery, and basically store a snapshot of the state of the subroutines being generated. With the aid of this snapshot, the error recovery can determine which type of operation was taking place when the error occurred. To aid this snapshot, when one of the mark op codes is compiled, the DEST parameter contains the ROW# and the LEFT parameter contains the COL#.

Internal representation

To sidestep the problem of allocating four separate areas for the four subroutines, the pseudo code is generated into one large linked list. The vector CODE is INTEGER*2. This vector contains the pseudo code. Each entry is from two to four entries long, and has the format:

WORD 1:   OPCODE + (64*NEXT).

WORD 2:   INDEX₃

WORD 3:   INDEX₂

WORD 4:   INDEX₁

where NEXT is the array index of the next instruction in the subroutine. If NEXT=Ø, the subroutine exits. Note that for operations requiring less than three addresses, the extra indices are not stored.

The array SUBS is a 2x4 array containing in its first entries the index of the first instruction of the particular subroutine. If this field is zero, an immediate exit is taken. The second element of the array is used to keep track of the index of the previous instruction generated for that

subroutine.

The array MAP is seven entries wide. It is here that the snapshots are stored when a MARK instruction is compiled. The snapshot contains the following seven items:

MAP(1,*)	OPCODE (2, 3 or 4)
MAP(2,*)	ROW#
MAP(3,*)	COL#
MAP(4,*)	MATIN1 current index
MAP(5,*)	MATIN2 current index
MAP(6,*)	MATIN3 current index
MAP(7,*)	MATMUL current index

In order to add new op codes to the routines, these steps are necessary:

- 1) include an entry for the op code in the table OPTYP indicating its addressing scheme
- 2) insert the operation into the computed GO TO in COMPIL
- 3) insert it into the GO TO in MATMUK
- 4) change the variable MAXOP to indicate the highest legal OP value

3.3.4 Interpreter Language Assembler (M. Wolfe)

General

I am currently writing an assembler for the interpreter language to be used in rewriting GRASS. The assembler's input is the program written in mnemonic code, and the ultimate output is the character string representation of the code in octal to be read into the PDP-8 which will interpret and execute the code.

First I will discuss some of the syntax of the language. The language

is composed of statements. Each statement has three parts:

- 1) label (optional)
- 2) instruction
- 3) operand(s) (sometimes)

All labels are optional, but when one is used, it must be a character string beginning with an alphabetic, composed of alphabetic and numerical characters, and must be followed by a colon (':').

The instruction field is necessary in all statements. A null statement, a blank card, is passed over.

The operand field is necessary in some instructions, optional in some, and not used in others. Most instructions which use an operand have a default operand, often zero, which is used if an operand is not specified.

Each statement is assumed to occupy one line (card) of the input. There are two ways to modify this convention. More than one statement may be on a single card if the statements are separated by semicolons(';'). The semicolon is read as the end of the current statement, and a new statement is assumed to start after the semicolon on the same card.

Also, a single statement may run over more than one card. When the current statement continues on the next card the programmer must put an asterisk as the last character on the current card. This asterisk may occur anywhere on the current card, but nothing following the asterisk is scanned, so any part of the statement must be before the asterisk or on the following card. A statement may be continued onto as many cards as desired.

Comments are enclosed in percent signs. A comment does not run over to the next card even if not closed by an ending percent sign. If a comment takes more than one card, each card must begin with a percent sign.

Each program is composed of an external procedure. Each procedure

must be compiled separately, and each must occupy no more than 4K of core, a limitation of the PDP-8.

Each external procedure is composed of internal procedures. The main routine is the first. Each internal procedure begins with a procedure declaration

name: PROC {(parameters)}.

Each ends with an END statement.

Assembler Workings

The assembler uses three symbol tables, one for each instruction field. The first of these, the instruction symbol table, is used to decode instructions. The label symbol table is used to store information about statement labels and procedure names. The variable symbol table is used to store global and local variables, constants, and pseudovariables which may be used in the program.

Each symbol table is made of three parts. The first is a character string representation of the elements of the symbols. The second is an integer array called TABLE which has four entries for each element. The first entry is a pointer to the character substring of CHARS representing that element. It is set to the subscript of the first character of the character string in CHARS of that element. The second entry contains the length the character string for that element. With these two entries we can retrieve the character string name for the entry with the SUBSTR function by

SUBSTR (CHARS, ENTRY1, ENTRY2).

The third and fourth entries are pointers to the left son and right son of this entry in a binary tree. This tree is sorted alphabetically, with the first entry being the root of the tree, and succeeding entries entered in the left or right subtrees, depending upon whether they are alphabetically

greater than or less than the root. The search procedure works similarly, but instead of entering a new element, if the searched for element is not found, the search fails.

Types of Instructions

Type

- 1 : No operand instructions. The operand field of the instruction is ignored. To assemble these instructions, the op code is inserted into the code and the stack is adjusted, if necessary.
- 2 : One operand, five bit, greater than zero instructions. These instructions have a constant operand which must be decoded and represented in five bits. This number, n , therefore must satisfy $0 \leq n \leq 31$. The instruction is one word long.
- 3 : One operand, five bit, sign magnitude instructions. Like type 2 instructions, these have a constant operand which must be decoded and represented in a sign magnitude number, n , of five bits, so that $-15 \leq n \leq 15$.
- 4 : One operand, five bit, two's complement instructions. Like types 2 and 3, these instructions have a constant operand, decoded into a number, n , which must be a five bit two's complement number, $-16 \leq n \leq 15$.

In types 2, 3, and 4, the operand and op code are combined into one word and inserted into the code, and the stack level is adjusted.
- 5 : One operand, seven bit, stack addressing instructions. These instructions can take either one or two words. The operand is the address of a stack level. In one word, this instruction can address 63 global stack levels and 32 local stack levels. With two words, this instruction can address 2048 global levels and 1024 local levels.

The operand field of this instruction is the last seven bits of the first word. One of these bits is the global bit; if the global bit is on, the operand refers to a global stack level. Another bit is the indirect bit. If the indirect bit is on, then the operand field refers to the address of the desired stack level.

If the desired level cannot be addressed in one word, then a second word is added, and the address is put into the second word of the instruction. Global stack levels are addressed by their absolute stack level, while local levels are addressed by the offset of the desired level from the current stack level.

To assemble this instruction, the operand is decoded. If it is a global level, then the proper level representation is put in the seven bit operand field, or the second word if it won't fit in seven bits. If it is a local level, then the offset from the current level is computed, and that is used in the seven bit operand field.

6 : One operand branch instructions. These instructions have one operand (a statement label) to which a branch is executed upon certain conditions. To assemble this instruction, a link to the symbol table entry of the operand label is generated and inserted into the code in the word following the op code. In pass 2 of the assembler, this link is translated into the address of the label, and a check is made to assure that the branch is made to a statement with the same stack level, so local stack level addresses are the same.

7 : One operand, text string instructions. The operand for these instructions is a text string. This text string is encoded into six bit ASC II, and stored two characters per word in the program.

The last five bits of the first word, which contains the op code, is used to tell the interpreter how many words follow the first word. The second word is used to tell how many characters are in the string. The operand for this instruction may also be a previously declared pseudo variable which will refer to a text string.

8 : Two operand, branch address and text string. This type is a combination of the above two types. The first operand, a branch address, is decoded and a link to the symbol table entry is put in the word following the op code. Then the text string is decoded, as in type 7, and inserted following the branch address.

9 : BRTBL (branch table). This instruction has a variable number of operands, each of which is a statement label. Each label is decoded and a link, as in type 6 instructions, is inserted into the code. The last seven bits of the op code are used to tell the interpreter how many labels are following the op code. The operand may also be a pseudo variable which has been previously defined to refer to a list of labels.

10 : TESTBR (test branch). This instruction has two operands, a constant and a branch address. The constant is decoded and put in the last seven bits of the op code. A link to the symbol table entry of the label is put in the second word, to be decoded in pass 2.

11 : SETGRID (set grid). This instruction has two variable length lists as input. Each list is composed of constants which are decoded and put into the code following the op code. The last seven bits of the op code are used to tell the interpreter how many words

of the instruction follow the op code. The second word of the instruction tells how many entries of the first word follow. Then the first list follows. The next word contains the number of entries in the second list, and is followed by the second list.

- 12 : LOAD. This instruction loads the operand onto the stack, a push operation. The load can be direct, indirect, or immediate. It can take one or two words. If the operand is a constant, then the constant is loaded immediately onto the stack. If the operand is a stack level, then the contents of that level are loaded onto the stack. The operand is put in the last seven bits of the op code, if it will fit; else it is put into the second word.
- 13 : Pop. The operand to this instruction is a constant. This is decoded and put in the last five bits of the op code. The instruction pops a certain number of levels from the stack, and the stack level must be adjusted.
- 14 : CALLI (call internal). This instruction calls an internal procedure. The operands are a list of parameters, each of which generate a load to load the address of the parameter. Then op code for CALLI is inserted into the program, and the address of the subroutine is put in the second word following the op code. The last five bits of the op code tell how many parameters were pushed onto the stack so that upon return from the subroutine, the stack is returned to its original level.
- 15 : CALLE (call external). This instruction calls an external procedure. It is assembled like CALLI.
- 16 : WAIT. This instruction halts execution of the program until certain conditions are met. The operands specify what conditions are being

waited on, and these are translated into a bit pattern in the word following the op code.

17,18 Not used.

19 : DGV (declare global variable). This instruction pushes a new level onto the stack. It requires a label, which is then defined as a global variable with address of that level. Because global stack variables must be at the bottom of the stack, this instruction must precede all others in the procedure.

20 : DSV (declare local variable). This instruction, like DGV, pushes a new level onto the stack, but defines its label as a local stack variable with address at that level. This instruction may occur anywhere in the procedure, but must occur before the variable is used.

21 : TEXT. This is a pseudo-instruction which requires a label and a text string operand. The label is defined as a pseudo-variable equivalent to the text string for use in type 7 and 8 instructions.

22 : GRID. This pseudo-instruction also requires a label and an operand, which will be a grid partition in the form of two lists like those used in the SETGRID instruction. The label is defined to be equivalent to this grid partition for use in SETGRID instructions.

23 : ADDRtbl. Like the above, this pseudo-instruction defines a label to be equivalent to the operand, which is a list of statement labels. It is used in the BRTBL instruction.

24 : EQU. This defines its label to be equivalent to a constant or to a stack level relative to the top of the stack. Like other pseudo-instructions, it requires a label and an operand.

25 : PROC. This defines a label to be a procedure name, and begins a

new procedure. It may be followed by a list of parameters which are defined as local stack levels. When this procedure is called, the addresses of the true operand should already be on the stack.

26 : END. This ends the current procedure.

Assembly Process

First, the card is scanned for the first token. The scanner has a look-ahead, so it recognizes a label by the colon. If the first token is a label, then it is saved and the card is scanned further.

When the instruction is found, then it is searched for in the symbol table. If it is found, then the entry is decoded as to op code, instruction type, and how the stack is affected. Then the assembler goes to assemble it according to the type of instruction it is.

If an operand is expected, then the card is scanned further until another token or the end of the statement, in the form of the end of the card or a semicolon, is found. The token is decoded, and checked to see that it is a valid operand. If no operand or an invalid operand is found, then a default operand is used. The assembled instruction is saved in an integer array. This array represents the 4K of core of the PDP-8. Every 128 words comprise a "page". When the instruction is put into the array, it must not run over a page boundary. This is important only in multiple-word instructions. If a label was defined for that statement, then it is defined at the location at which the instruction is inserted. Then the process continues for the next instruction.

Addresses are assembled in the second pass.

In the first pass, all branch addresses are stored in the integer array as special pointers to symbol table entries. In the second pass, these pointers are traced and the actual addresses are inserted. The stack levels

are checked for compatibility.

Then the code is outputted into a file so it can be read into the PDP-8 for interpretation and execution.

3.4 Satellite Processor Software and Hardware

3.4.1 CTKTST - Backpanel Wiring Test Program (R. Whyte)

CTKTST was written to allow the back panel wiring of sample computek terminals to be tested against a standard terminal with a wiring list resulting. The software merely generates and interprets data used by the testing hardware. Along with this it provides keyboard control, hard copy output of lists, and a file of the standard terminal connections.

Using CTKTST

The software program consists of two parts, a generation state in which a standard terminal is 'read' and a wiring list is generated and an interrogation state in which a sample terminal is tested and a discrepancy list is generated. Since the computek is composed of twenty-six, eighty pin, plug in boards with wiring connections both within a single board and between two separate boards, the software program runs in two modes, a self sensing one in which intraboard connections are read or tested and an alternate-sense one in which interboard connections are read or tested. These two nodes are reflected in the use of the two testing boards which are substitute plugged into various board positions in the computek. By employing the software program and the hardware testing boards properly, inconsistencies in a terminal can be discovered without damage to the hardware of the terminal.

The proper method for running and using the program is as follows. To load the test routine the user must type on the PDP-8 console:

. R CTKTST

which causes the OS/8 operating system to locate and load the test program.

Once the program is loaded and is in control it will respond by typing an asterisk at the console. At this point the user selects the devices and names with which the program will associate. The general form of the command line is as follows:

output device: < file device: file name/function

where

output device: is the device, either TTY for the PDP-8 console or LPT for the line printer, on which pin connections are printed if in generate state or pin connection discrepancies are printed if in test state

file device: is the device on which the pin connection table is stored if in generate state or the pin connection table is to be found if in test state.
(DTA_, DISK)

file name: is the name of the file in which the pin connection table is stored if in generate state or the pin connection table is to be found if in test state (FILE, PA...)

function: is the state in which the program should run. G is for generate state and T is for test state.

NOTE: if any of the above are not specified a default value is substituted. The default values are:

TTY: < SYS: CTKLST/G

The program now begins to build the wire connection table if generating, or gets the already generated table from the device if testing. When this is accomplished it will type:

DRIVE = A without a carriage return or line feed. A is the card which the program expects to be driven. If this is not the card the user wishes to drive one merely types the letter of the card to be driven. In both cases, the drive test card should be positioned in the terminal to be tested in the

corresponding slot and the carriage return typed. The program then types:
SENSE = A again without a carriage return or line feed. The same procedure
should be followed as for the drive card only this time with the sense card.
In the case of self-sensing (i.e., drive and sense cards specified are the
same) the sense card is not placed in the terminal. When carriage return is
now typed the program begins by driving the first pin on the drive card and
sensing the second through last pins on the sense card if self-sensing, or
first to last if alternate-sensing. Output responses are now determined by
the state of the program.

GENERATE STATE - If no function or G was specified the program will print on
the output device any pins which are connected. The output might look like:

```
DRIVE PIN = 1      SENSE PIN = 2
                                     5
                                     :
DRIVE PIN = 2      SENSE PIN = 3
DRIVE PIN = 60     SENSE PIN = 68
                                     72
```

when the entire set of drive and sense pin combinations have been generated
the program will then type the next set of expected test cards

```
DRIVE = A
SENSE = B
```

and so on until all the boards have been tested. At this point the completed
table is output to the specified device under the specified file name and the
program returns to the command decoder for the specification of a new command
line. The user would probably then wish to begin the test of another terminal.
TEST STATE - If T was specified as the function the program will be in test
state, and will print any pin connection discrepancies. The output could have

two forms depending upon the irregularity. The output might look like:

1   OUGHT TO BE CONNECTED TO 6

7   SHOULD NOT CONNECT TO 10

and so on. As in the generate state when a card pair is exhausted, a new expected pair is printed and the process continues. When all card pairs have been tested the program returns control to the OS/8 system for the specification of a new program.

The user also has the ability to stop the program action at any time by typing various control characters. These control characters and the resulting action are as follows:

↑C - exit program and return to 0./8

↑P - exit to commod decoder to specify new commod line

↑O - stop current test to specify a new card pair

rubout - rub out last character typed

↑U - rub out last line typed

↑T - set output device to TTY

↑V - set output device to LPT

Various errors can result using CTKTST. Some of these are due to OS/8 system faults while others are the result of a user fault. When CTKTST detects an error it outputs an error message on the console and exits control to the OS/8 system. The error messages are as follows:

DEVICE ERROR   occurs when a parity error is detected on I/O

LOOKUP ERROR   occurs when the specified table file does not exist on the
                  specified device

ILLEGAL CARD
SPECIFICATION   occurs when a request is made to test a card which is not in
                  the computek

GENERATE TABLE
EXCEEDS SPACE   occurs when the generation table is too large. Recovery

could be made by generating two tables, each half of the overall one.

CTKTST - Software Details

Basically CTKTST consists of four nested loops which increment counters to test all possible unique drive card (DCARD), sense card (SCARD), drive pin (DRVPIN), sense pin (SNSPIN) combinations. This is prefaced by the necessary OS/8 system software calls to interpret the user typed command line, load device handlers, and initialize other program pointers.

For each choice of DCARD, SCARD, DRVPIN and SNSPIN a hardware test is generated and interpreted in light of the state of the program. The hardware test is accomplished by setting certain bits in the accumulator. The hardware test interprets the accumulator bits in the following way:

where

PIN NUMBER is the internal identifier for the pin to which the test is to be applied.

ALTERNATE SENSE BIT is set if the sense and drive cards are not the same.

DRIVE BIT is set if the hardware function is to drive the numbered pin.

When the drive bit = 0 the hardware returns bit 11 set if there is a connection to the previously driven pin.

Once the hardware test is made the result is interpreted in light of

the state of the program. In GENERATE STATE a table of pin connections is produced in core with the following format:

PIN CONNECTION TABLE		
INTERNAL DRIVE CARD NUMBER	INTERNAL SENSE CARD NUMBER	← TCARD
DRIVE PIN NUMBER		← DPIN
COUNT OF CONNECTIONS		← PCOUNT
SENSE PIN NUMBER OF FIRST CONNECTION		← SPIN
SENSE PIN NUMBER OF SECOND CONNECTION		↓
⋮		
INTERNAL DRIVE CARD NUMBER		
⋮		
END MARKER		
3	7	7 7

The table values are referenced by the pointers indicated where SPIN actually points to the next available entry in the table. When a drive pin and a sense pin combination results in ACC11 bit being set by the hardware test the following operation is performed on the table.

In this way the table only reflects the occurrence of a connection between a given card pair. Space is therefore saved by not recording even the card names for cards which have no inter-connections.

The pin pair which connect are then output to the appropriate device via some output routines which convert internal pin numbers to printing characters via a table lookup. When all the cards have been done, the table is output to the specified device using OS/8 system software device handlers.

In test state the table is read in from the device on which it is stored and pointers are initialized to the top of it. Since the table is sparse and contains only connections and no other information, it is necessary to perform the following test routine each time a hardware test is made. In this way instances of missing connections will be detected. The testing routine looks like this:

In this way the test routine pointers are never behind the actual parameters and will be incremented when the actual parameters are reached. When all the cards are tested the program returns to the OS/8 system.

One final feature is included in the program. If the card values specified by the user do not correspond to those which the program expects a scan is made of the table to set the table pointers properly. This scan is accomplished by setting the program to the test state and disabling all output. The program then performs the test loop until the proper card value is pointed to in the table or the one pointed to is larger than the desired value. The state is then returned to the original one, output is enabled and the program continues. This method while slow is sufficient and involves no more programming effort.

3.4.2 PDP-8 Software (N. Hennegan)

PL

This is an OS-8 utility program to communicate with the IBM 360 filing system. Basically this routine allows the user to send, retrieve and store OS-8 source files in PLORTS. This alleviates the problem of only one person being able to edit PDP-8 source code at a time. Upon entry, the user is in direct communication with plorts. By typing:

CTRL/C - The channel is disabled and control returns to the user.

CTRL/P - The command decoder is brought in enabling the user to specify either an OS-8 file to receive input or OS-8 files to be output. When an input file is specified (by a trailing '<'), the currently opened plorts file is loaded into the specified input file. If the PLORT line numbers are to be included, the '/L' option must be used. For example:

* LPT: </L

will cause the file to be output on the line printer with the PLORT line numbers.

If files are to be output to PLORTS (i.e., entered in PLORTS) the user must first open a file and get into the ENTER mode (see PLORTS MANUAL). The user then types CTRL/P and then the name or names of the files to be entered in PLORTS. Upon receipt of a CR/LF, the transfer is complete but the user is still in enter mode! The user may use the '/E' option which sends the string 'EXITTT' after the last file which exits the user from the ENTER mode. For example:

* DTAØ: FILE1/E

will cause FILE1 on DTAØ to be entered into the current PLORT file and an 'EXITTT' to be sent at the conclusion of the transfer. (Note the default extension is .PA.)

CTRL/V - Switch output device to the line printer and do a carriage return.

(i.e., by typing a LN+V, the first 31 lines of the files will be listed on the LPT. A 'CR' will output the next 31 lines.)

CTRL/T - Switch the output device to the TTY and do a local CR/LF.

CTRL/L - Send a 'EXITTT' to PLORTS

CTRL/I - (Tab) Tab to next tab stop - every eighth column.

CTRL/O - Stop further output until keyboard input.

ALTMODE- Same as CTRL/O

CTRL/U - Do a local deletion of the current line in TTY buffer.

CTRL/W - Same as CTRL/U

RUBOUT - Delete last typed character.

SHFT/N - Same as RUBOUT

LINE FEED - Retype the current line to the present position as it appears in the PLORTS input buffer.

I encountered three major problems in writing this routine. The first is that the supervisor for the channel expects 6-bit packed ASCII characters. Two control characters CTRL/Z (232) and CTRL/L (214) which are not included in the trimmed ASCII set are used extensively in the OS-8 filing structure. The CTRL/Z is used to signify an end of file and the CTRL/L is used by the paging editor to signify the end of a block of text. To overcome this difficulty and still allow the user to enter the same information through plorts the two characters are converted to the text string 'ENDOFFILE' and 'FORMFEED' when sent to the filing system and reconverted when received. Hence when creating an OS-8 source file in PLORTS, the user should insert a 'FORMFEED' in at least every 100 line of text and end the file with the string 'ENDOFFILE'. This last string is particularly important since the program expects the string and will loop until it is received.

Another problem encountered with the channel supervisor is that a line greater than 64 characters is sent through the channel as two lines. This caused all sorts of havoc with the assembler because comments were showing up in the instruction fields. This problem could not be easily handled with the program in the PDP-8 so the command '#LONG' and '#D LONG' have been added to the channel supervisor. This allows text lines of length greater than 64 characters to be sent as one line. Upon entry and before each file transfer, the string '#LONG' is sent to the supervisor. The string '#D LONG' which negates #LONG is sent to the supervisor before returning to the OS/8 monitor.

The third problem deals with sending text line of a file to PLORTS at too great of a rate. The filing system PLORTS was designed for input from fairly low speed terminals such as teletypes, while the channel is capable of

transmissions at about one-half a million baud. PLORTS will spool any command or text that it cannot immediately accept so that effective transmissions are reduced once we exceed the point at which PLORTS can accept the text line immediately. A further problem occurs when the user disk allocation becomes filled. When this happens, the filing system returns the message 'DISK FULL' which is also buffered on disk since the channel is half duplex and is busy sending messages to PLORTS. We then get a backlog of spooled error messages.

After a bit of trial and error I have found that pausing about 60 milli seconds between each line transmitted to PLORTS and concurrently testing for any messages from PLORTS gives satisfactory results at times when the filing system is not saturated. Text line transmissions run close to 800 per minute. Very rarely though, the filing system may become saturated and the message "WAIT" or "DISK FULL" is received. If this occurs the user may decide to just try again or he can set the low 5 bits of the switch register to additional delay increments of about 20 m.s., i.e. setting the low 5 bits of the switch register to 3 will delay each line transmission an additional 60 m.s.

3.4.3 PDP8 Party-line AC Expander (A. Whaley, W. Tam)

Various peripheral devices had been connected to the PDP8 and most of these communicate via the accumulator. The non-uniform interfacing of these devices, both logically and electrically, had caused considerable problems in trouble-shooting or modifying the system. As more devices are expected to be added, standardization of interfaces becomes a must. Also means were sought to minimize the number of connecting wires to the accumulator.

On close examination of most peripheral devices we found that the following signals were most frequently used:

- (i) up to 12 bits of accumulator content to the device
- (ii) up to 12 bits of data to be loaded into AC
- (iii) 3 IOP pulses
- (iv) the lower 9 bits of the memory buffer (MB)
- (v) load accumulator (LDAC), clear accumulator CLAC, SKIP and INTERRUPT signals.

The signal lines in (i) and (ii) are combined using half-duplex interfacing (party-line). The basic circuit for one accumulator bit is shown in Figure 3. The direction of signal flow is controlled by the LDAC signal. Normally LDAC is at logical zero for all devices. The open collector gates in the devices are turned off, the buffer gate (also open collector) is enabled. The content of the AC is then available to all devices. The open collector buffer gate can drive up to 30 inputs. When a device wishes to load the accumulator, it will "pull down" on the $\overline{\text{LDAC}}$ line (LDAC goes to logic 1). The open collector gate in the device interface is turned on, and the buffer gate is turned off. Thus the AC signal line is "turned around". The data bit(s) from the device is loaded into the AC via an OR gate, which is controlled by a delayed $\overline{\text{LDAC}}$. This is to ensure that the signal line settles down before information is loaded into the AC.

Using this scheme, the number of connections to the AC bits are reduced by half. Standardization is achieved as well as inter-device independence. Devices can be added or removed without locking out the AC inputs.

As for the MB and IOP lines, a simple open collector buffer capable of driving 30 inputs is used for each line. For the CLAC, SKIP, INT signals, each device is to provide the signal in complemented form via an open collector output gate. The common load resistor for all such gates is on the bus. Thus

devices generating these signals can be plugged in or pulled out without affecting the PDP8 and other devices.

Figure 3

	1	2	3	4	5	6
A						
B						
C						
D	7438	7438	7438	7400		

PDP-8 PARTY LINE

BUFFERED MB & CONTROLS

A. WHALEY

PSP

7/30/74

2383

p. 2 of 2

7-4

3.4.4 Computek Wiring Diagnostic Device (W. Tam)

Although this device was designed to check computek wiring errors, it can be easily modified for other similar purposes. The device is actually an addressable "make or break" detector. It communicates with the PDP8 via the accumulator panel. The input and output of the AC has the following formats.

The device can be divided into three parts:

- (i) pin driver
- (ii) self sensor (or local sensor)
- (iii) alternate sensor

COMPUTEK WIRING DIAGNOSTIC DEVICE
BLOCK DIAGRAM

Each of these parts has a 7 bit pin number register which is selectively loaded. The pin driver uses two levels of BCD decoder/drivers to drive one out of 80 pins to ground according to the content of the driver register. The sensors are one-out-of-80 data selectors that can sense any one of the 80 lines specified by the content of the 7 bit sensor registers. The difference between the self and alternate sensors is that the 80 pins which the self sensor senses is the same 80 pins that the driver drives, while to alternate sensor is mounted on another 80 pin PC connector card.

The accumulator is used both to send instructions to the device and read in the result of the last sense instruction. The device has a 44 pin card connector which plugs into the accumulator panel. When the PDP8 executes an 10T 6461, the strobe cause the lower 7 bits of AC to be loaded into either the driver or one of the sensor registers, depending on bits 3 and 4 of AC. If bit AC3 is 1, the 7 bit pin number will be loaded into the driver register and the corresponding pin will be driven. If AC3 is zero, either the self sensor register or alternate sensor register will be loaded according to AC4. The result of the sense will then be available as bit 11 of AC input (1 if a driven line is sensed). This information will be loaded into the AC when the PDP8 executes 10T 6466.

A micro switch was mounted on the driver card. Whenever the switch is pressed, bit 0 of the AC input will be set to 1, and will be loaded into the AC nexttime the PDP8 executes a read 10T (6466). This bit is used only to convey extra control information to the program controlling the device, and does not cause any hardware interrupts.

To use the device, the PDP8 runs a program that exhaustively drive and sense the condition (open or connected) between every pair of pins, and compares the result with a list of connections stored in the program. Any

discrepancy can then be printed out and the wiring error corrected.

3.4.5 Accumulator Panel (W. Tam)

Constructed as one of the standardized devices using the party-line feature, the accumulator panel allows temporary connections for accumulator inputs and outputs. The panel provides 12 bits each of inputs to and outputs from the AC, a strobe signal and a T5V supply. The operation of the panel is controlled by two instructions. 10T 6461 will load the output latches from AC as well as give 10P1 as strobe. 10T 6466 will load into AC the inputs from the panel. Connections can be made either via the binding posts or thru a 44 pin connector socket.

U. S. ATOMIC ENERGY COMMISSION
UNIVERSITY-TYPE CONTRACTOR'S RECOMMENDATION FOR
DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

1. AEC REPORT NO.

C00-2383-0015

2. TITLE

QPR-74-3

3. TYPE OF DOCUMENT (Check one):

☒ a. Scientific and technical report

☐ b. Conference paper not to be published in a journal:

Title of conference _____

Date of conference _____

Exact location of conference _____

Sponsoring organization _____

☐ c. Other (Specify) _____

4. RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

☒ a. AEC's normal announcement and distribution procedures may be followed.

☐ b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.

☐ c. Make no announcement or distribution.

5. REASON FOR RECOMMENDED RESTRICTIONS:

6. SUBMITTED BY: NAME AND POSITION (Please print or type)

C. W. Gear

Professor and Principal Investigator

Organization

University of Illinois

Department of Computer Science

Urbana, Illinois

Signature

Date

December, 1974

FOR AEC USE ONLY

7. AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

8. PATENT CLEARANCE:

☐ a. AEC patent clearance has been granted by responsible AEC patent group.

☐ b. Report has been sent to responsible AEC patent group for clearance.

☐ c. Patent clearance not required.

4. SWITCHING THEORY AND LOGICAL DESIGN

(Supported in part by the National Science Foundation under Grant Number U.S. NSF-GF-40221.)

During the summer we made good progress on our work. R. Cutler completed many subroutines of Tison's methods. Professor K. Kinoshita of Osaka University left here at the end of July. Before his departure we finished major part of our joint exposition paper on Tison's methods. K. C. Hu continued assembling transduction method programs into a system, using Job Control Language.

H. C. Lai proved the optimality of the parity function network for an arbitrary number of variables which used to be a conjecture in our group. J. Culliney was away for a major part of the summer for the military obligation. B. Plangsiri started writing of his master thesis on the design of NOR networks under fan-in and fan-out restrictions by the transduction methods. M. Malouf made some progress on his master thesis on the design of optimal MOS networks by integer programming.

S. MUROGA

Work continued on my dissertation. Also time was spent preparing several reports on NOR network transduction procedures for publication as departmental reports.

(J. Culliney)

Work continues on the programs to implement Pierre Tison's Algorithms. Preliminary results are optimistic. In a recent run, all the prime implicants for the 402 NP equivalence class functions of 4 variables (or less) were found in less than 90 seconds 360/75 processor time. Work is nearing completion on an exposition paper explaining Tison's method.

(Robert Cutler)

The NETTRA system which was designed last quarter was being debugged this quarter. There is only one subsystem which has not been completely debugged as yet. Ten 5-variable functions are being used to test the system: the computation time that each subsystem spends and the best cost that each subsystem achieves are recorded so that we can determine reasonable ways of setting default values for input parameters.

More functions will be used to test the efficiency of the system and the default values for input variables will be set soon.

(K. C. Hu)

Theoretical studies on several problems in logic design were continued. It was proved, that NOR (or NAND) networks for parity functions of n variables require at least $3(n-1)+1$ gates. Since a method to construct these networks with $3(n-1)+1$ gates has been shown, the networks constructed by this method have been proved to be minimal networks for parity functions.

(H. C. Lai)

LIST OF PUBLICATIONS

Lai, H. C., Nakagawa, T., and Muroga, S., "Reducing Check Technique for Designing Optimal Networks by Branch-And-Bound Method," International Journal of Computer and Information Science, Sept. 1974, pp. 251-271.

5. MACHINE AND SOFTWARE ORGANIZATION STUDIES

(Supported in part by the National Science Foundation under Grant Number
US NSF GJ-36936)

The following is a collection of related work aimed at improved designs for computer and software systems. We are interested in parallel and pipeline processors, small primary memories, effective use of rotating memories, and some questions concerning user languages for problems including typical FORTRAN type calculations, simulation languages, and a variety of file processing problems.

(D. Kuck, D. Lawrie, J. Liu and A. Sameh)

5.1 Algorithm Bounds - (A. Sameh)

We have developed direct parallel algorithms for the solution of the Poisson and biharmonic equations on the unit square, using a number of arithmetic units in parallel. Superimposing a square grid over the unit square ($h=1/n$), we show that a finite difference approximation to the Poisson equation can be solved in $O(\log_2 n)$ time steps using $O(n^2)$ processors in parallel. This holds for the Dirichlet, Neumann, and periodic boundary conditions. Similarly, we show that a finite difference approximation to the biharmonic equation can be solved in $O(n)$ steps with $O(n^3)$ processors.

In solving systems of linear equations $Ax=f$ where A is dense, sequential algorithms require $O(n^3)$ steps. We have considered parallel algorithms for both the LDU and QR factorizations of A . We show that for positive-definite symmetric matrices the LDL^t factorization may be computed in $O(n)$ steps using $O(n^2)$ processors. If A is not positive-definite, pivoting is required and the process is of $O(n \log n)$ steps. The orthogonal

factorization $A=QR$ in parallel has been achieved by square root-free Givens' transformations in $O(n)$ steps with $O(n^2)$ processors. Hence, this factorization is preferred when pivoting is necessary for a stable LDU factorization.

Givens' reduction can also be applied to tridiagonal matrices. We show that the factorization can be obtained in $O(\log n)$ steps using $O(n)$ processors. This result is applied directly to the QR-algorithm for finding the eigenvalues of symmetric tridiagonal matrices so that one iteration may be performed in $O(\log n)$ steps without any square-root evaluations.

5.2 PROGRAM ANALYSIS

5.2.1 Speedup of Iterative Programs - (S. C. Chen)

In this quarter, we extended the results obtained from the previous study of linear recurrence relations to ordinary program loops. A theoretical speedup spectrum of different types of loops in terms of the problem size (sequential computation time) is established. We can also bound the times of some loop structures with known computation patterns (dependence graphs) and certain measurable parameters, with which the minimum speedup on a multiprocessing system of a particular class of loop computations (and hence the algorithms they represent) can be estimated.

A report to summarize all these initial results is under preparation, and the research is continuing on some application-problems in numeric algorithms and compiling techniques for parallel machines.

5.2.2 IF Analysis - (R. Towle)

Work on more accurate tests for data dependency within DO loops has continued. Several theorems have been proven. The major theorem is

Thm Given two statements S_ℓ and S_m ($m \neq \ell$) in the same DO loop with control set I where S_ℓ has a_{i-p} on the left-hand side and S_m has a_{i-j} on the right-hand side. Let the index set at S_ℓ be AIS d_1, b_1 (i.e., of the form $d_1x - b_1$) and the index set at S_m be AIS d_2, b_2 . There is a data dependency from S_ℓ to S_m if and only if $(d_1, d_2) \mid (b_1 - b_2 + p - j)$ and $j \geq p$.

5.2.3 Machine Simulator - (D. Y. Chang and K. Y. Wen)

The simulator we wrote last summer seems too primitive. So in this quarter, work has been done to modify the data structure and to improve the algorithms used in our simulator. Besides, we added new algorithms that simulate fixed-head disk operation, timing the DO loops. Eventually, we want our simulator to simulate all kinds of resources like moving head disk, tape, etc.

We expect to finish coding this simulator and run some trivial hand-coded sample programs in a few months.

5.2.4 Alignment Network Study

Various types of alignment networks are being studied. Some of the more important networks under study are the Batcher Network, Beneš Network and Lawrie's shuffle network. The main areas of explorations are the functional and topological comparisons of the various networks. Simplifications in control algorithms of these networks are also being attempted.

5.3 FILE PROCESSING

5.3.1 File Processing Software - (A. Cheng and B. Hurley)

During the last quarter, efforts centered on further refinements of simulation results for a single user with head-per-track disks as bulk storage. Plans to continue the simulations of a specialized information

retrieval computer using different types of bulk storage such as movable arm disks and shift registers are now being discussed.

The last quarter also saw the beginning of work on a program to do merging of large postings lists for our Information Retrieval System. This code will outdate the old version by allowing any two lists to be merged regardless of their size. When this project is completed, a new data base consisting of business abstracts can be added to our system. Further, time was devoted to managing and editing the documents required for the future IRSYS information retrieval system.

5.3.2 Inverted File Retrieval System - (M. Milner, K. Morgan
and J. Rinewalt)

During the last quarter, some maintenance has been performed on the current software, while work has begun on a new version that will allow a restricted macro facility and a browse mode for viewing documents. Various improvements in disk space allocation algorithms and directory organization will be included in this version. Work will continue on the new version during the next quarter.

Four hundred fifty business abstracts were made ready for addition to the system. They will be added as soon as the merge routine is modified to handle large data bases.

Work on the multi-user system has been delayed by other commitments. Design is, however, nearly completed and coding will begin in October. The core of a multiuser system should be running by mid-December.

5.3.3 An Approach to the Table Manipulator - (I. Hueck)

A set of meaningful operations between tables has been studied and

a set of commands for this purpose has been designed. The design of a general organization of the system has been completed. Special attention is being given to the following areas:

- a) Expandability of the system
- b) Compatibility with a bibliographic information retrieval system
- c) Comformability of the system user.

Future efforts shall be concentrated in the applicability of such a system to the fulfillment of actual necessities in the fields of industry and research centers.

5.3.4 Floppy Disk - (P. Krabbe)

We received a Model 74 Diskette Drive from ORBIS SYSTEMS, INC., during the month of August. The system has a capacity of 41666 bits per track (77 tracks total) unformatted and a transfer rate of 250 kilobits/second. Track-to-track access time is 6ms with 24ms settling time. Head load time is 30ms. The logical design for the controller is in progress.

5.3.5 Hardware Support - (J. Bengtson and L. Hollaar)

Design of an experimental merge processor for the PDP-11/40 based information retrieval system has been completed, and construction will begin when the necessary components arrive. The processor consists of three wirewrap boards, each containing about eighty TTL integrated circuits, mounted on a single DEC system unit within the existing processor cabinet.

As a prelude to the installation of the merge unit, the Datacraft memory was rewired to make it two separate 24K by 16-bit memories. Currently, these are connected together with a jumper module to allow operation as a 48K

by 16-bit memory with two-way interleaving. However, when the merge processor is installed the memory will be reconfigured to 24K by 32-bit to match the word length of a postings entry; when the memory is addressed by the UNIBUS, the switch section of the merge processor alters this connection to the standard 48K by 16-bit configuration.

The unit is capable of merging two input lists into an output list at memory speeds, allowing an increase in speedup over the conventional software by a factor of 15 to 25. If the memory used were faster, simple design modifications would increase the speed increase to 100.

Each input item is checked against a user specified mask to insure that the item is contained in the proper context. If it passes this test, it is held for comparison against the current item from the other input list. The result of this comparison selects commands to fetch new input items, write an output item, and select a document number from an input item and create tag and count fields for the output item. This programming allows great generality in specifying the desired function, providing many different operations in addition to the standard OR, AND, or AND NOT operations.

5.3.6 High Speed Secondary Memory - (D. Johnson)

To investigate the problems of modifying a disk drive so as to permit a multiplicity of heads to be actively processing data at a single time, plans were begun to incorporate a two active head system on the Century Data Model 114 Disk Drive.

The electronics of the 114 Disk Drive is such that a preamplification stage already exists for each set of 10 heads and, therefore, to

activate a second head, it is only necessary to duplicate the Read Amplifier, Crossover Detector, and Write Driver stages in the read/write subsystem of the disk drive.

Thus, during this quarter, the Read Amplifier, Crossover Detector, and Write Driver PC boards, along with a PCB rack and some technical manuals, were purchased from California Computer Products, Inc.

The foremost difficulty foreseen in the installation of a two active head system in the 114 will be the proper management of inter-head data skew and coordination.

Publications and Theses:

- D. E. Gold and D. J. Kuck, "Implementation Considerations for Masking Rotational Latency by Dynamic Disk Allocation," International Workshop on Modelling and Evaluation Computer Architectures and Networks, Aug. 1974, IRIA, Paris.
- D. Kuck and Y. Muraoka, "Bounds on the Parallel Evaluation of Arithmetic Expressions Using Associativity and Commutativity," Proceedings of the Seventh Annual Princeton Conference on Information Sciences and Systems, pp. 161-168, Mar. 1973; also appears in Acta Informatica, Vol. 3, Fasc. 3, pp. 203-216, 1974.
- W. H. Stellhorn, "An Experimental Information Retrieval System, Univ. of Illinois at Urbana-Champaign, Dept. of Computer Science Rpt. No. 657, July 1974. (NSF-OCA-GJ-36936-000005).
- K. Y. Wen, "Machine Parameter Deduction by Program Analysis," M.S. thesis, Univ. of Illinois at Urbana-Champaign, Dept. of Computer Science Rpt. No. 636, Aug. 1974. (NSF-OCA-GJ-36936-000002).

6. ASPECTS OF COMBINATORIAL COMPUTING

(Supported in part by the National Science Foundation under Grant No. US NSF GJ-41538)

1. Backtrack programming (J. Bitner and E. M. Reingold)

Research is being carried out in the use of macros to speed up backtrack programs. The following four problems were considered:

(1) The queen's problem: In how many different ways can n non-attacking queens be placed on an $n \times n$ chessboard? We are interested in both the number of solutions that are inequivalent under rotations and reflections, and the total number of solutions. A program solved the $n=15$ case (45,752 inequivalent solutions and 365,596 total) in 25 minutes. An earlier non-macro program required three hours. The $n=16$ case (1,846,955 inequivalent and 14,772,512 total), which was previously unsolved, was completed in 168 minutes.

(2) A pentomino problem: For what n can a $5n \times 12$ region be tiled by the pentomino ? The smallest known tiling was for $n=16$, due to Klarner. A program using macros took seven minutes to find a tiling for $n=10$ and to demonstrate by exhaustive search that no smaller tilings exist. An examination of this solution provided tilings for all $n \geq 10$.

(3) Squaring the square: Can a 70×70 square be tiled with square tiles of sizes 1 through 24? A program was written that searched through all possibilities and found no solutions in 16 minutes. We are now investigating the problem of using pieces to cover as much of the 70×70 area as possible. Since there is a known solution that leaves 49 squares uncovered, a program is being

written that will show this is the smallest area possible, or find a smaller one.

(4) Difference preserving codes: A difference preserving (DP) code of threshold t is a sequence of binary codewords $(a_1, \dots, a_k)$ each N bits long such that:

$$(1) \text{ for } |i-j| \leq t \quad H(a_i, a_j) = |i-j|$$

$$(2) \text{ for } |i-j| > t \quad H(a_i, a_j) > t$$

The problem is to find the longest DP code of threshold 1 for a given N . A program was written which found the longest 6-bit code (27 codewords long) in three seconds. A non-macro program required 42 seconds. The 7-bit code is currently beyond the program's capacity. It would take an estimated 140 hours to complete the examination of the search tree of 10^{10} nodes.

In the two cases where earlier non-macro programs existed, the use of macros increased program speed by factors of five to ten times. Indeed, this proved to be a valuable backtrack programming technique.

2. A Recognition Algorithm for the Intersection Graphs of Directed Paths in Directed Trees (F. Gavril)

Consider a finite family of non-empty sets. The intersection graph of this family is obtained by representing each set of a vertex and by connecting two vertices with an edge if the corresponding sets intersect. The intersection graph of a finite family of directed paths in a directed tree is called a directed path graph. We discovered an efficient algorithm to determine whether a given graph is a directed path graph. If the

graph has n vertices, the algorithm requires $O(n^4)$ steps. Also, we prove that a graph is a proper directed path graph if and only if it is a directed path graph.

3. Adaptive Merging by Parallel Disjoint Comparisons (F. Gavril)

We studied an adaptive algorithm for merging two linearly ordered sets A, B , $|A| = m$, $|B| = n$, $m \leq n$, using a constant number p of parallel processors which perform disjoint comparisons, i.e., no element is accessed in the same time by more than one processor. The time required by this algorithm is approximately $\log_2 m + \frac{3m}{p} + \frac{m}{p} \log_2 \frac{n}{2m}$ which clearly is asymptotically optimal. This time compares favorably with the non-adaptive merging algorithms using a constant number p of comparators, since it was proved by A. and F. Yao that in this case the optimal algorithm requires at least $\frac{n}{2p} \log_2 m$ units of time.

4. Lubell's Inequality in a Poset (H. P. Tsao)

Let P be a poset with $n+1$ levels $P_0, P_1, \dots, P_n$ such that

- i) each element of P is comparable with at least one element in every other level;
- ii) all chains connecting two elements of P have the same length;

$$\text{iii) } \frac{|P_{i+1}|}{|P_i|} = \min \left\{ \frac{|P_i(B)|}{|B|} : B \subset P_i \right\} \text{ for all } i, \text{ where } P_i$$

is the projection of P_i onto P_{i+1} .

Boolean algebras, the lattices of divisors of integers and modular geometric lattices all enjoy the above property. We have proved:

Theorem. For a fixed $l(1 \leq l \leq n)$, every family F in P , containing chains of length not exceeding l , satisfies the inequality

$$\sum_{i=0}^n \frac{|F \cap P_i|}{|P_i|} \leq l,$$

which we call Lubell's inequality.

5. The Logarithmic Concavities of the Binomial Coefficient Type Numbers (H. P. Tsao)

A sequence $\{S_j\}$ of real numbers is said to be strongly logarithmically concave (SLC) if $S_{j-1}S_{j+1} < (S_j)^2$ for all j . An array (S_j^n) $0 \leq j \leq n$ of positive numbers, attached with a non-decreasing LC sequence $\{\lambda_j\}_j$ of positive numbers, is said to be a binomial coefficient type (BCT) array if $S_j^{n+1} = S_{j-1}^n + \lambda_j S_j^n$ for all n and j , $0 \leq j \leq n$, where S_j^n is defined to be zero if $j < 0$ or $j > n$. For instance, binomial coefficients, Stirling numbers of the second kind, and Gaussian coefficients are all BCT. We proved the following theorem:

Theorem. If a BCT array (S_j^n) satisfies the initial condition $S_0^2 S_2^2 < (S_1^2)^2$, then both the sequences $\{S_j^n\}_j$ and $\{S_j^n\}_n$ are SLC. Moreover, if j_n (resp. γ_n) is the least integer such that $S_{j_n}^n$ (resp. $S_{\gamma_n}^{n-\gamma_n}$) is maximum, then $j_{n+1} = j_n$ or $j_n + 1$ (resp. $\gamma_{n+1} = \gamma_n$ or $\gamma_n + 1$).

6. Evaluation of Powers (A. C. Yao)

It is shown that for any set of positive integers $\{n_1, n_2, \dots, n_p\}$, there exists a procedure which computes $\{x^{n_1}, x^{n_2}, \dots, x^{n_p}\}$ for any input x in less than

$\lg N + C \cdot \sum_{i=1}^P (\lg N_i / \lg \lg N_i + 2))$ multiplications for some constant c , where $N = \max\{N_i\}$. This solves an open problem in Knuth (The Art of Computer Programming, Vol. III) and generalizes Brauer's theorem on addition chains.

7. Random 3-2 Trees (A. C. Yao)

It is shown that $\bar{n}(N)$, the average number of nodes in an N -key random 3-2 tree, satisfies the inequality $0.70N - \epsilon < \bar{n}(N) < 0.79N + \epsilon$. A similar analysis is done for general B-trees. It is shown that storage utilization is essentially $\lg 2 \approx 69\%$ for B-trees of high orders.

8. Merging Networks (A. C. Yao and F. F. Yao)

Let $M(m, n)$ be the minimum number of comparators needed in an (m, n) -merging network. It is shown that $M(m, n) \geq n(\lg(m+1))/2$, which implies that Batcher's merging networks are optimal up to a factor of $2 + \epsilon$ for almost all values of m and n . It is also proved that $M(2, n) = \lceil 3n/2 \rceil$, and that $r_m = \lim_{n \rightarrow \infty} M(m, n)/n$ exists. r_m is determined to within 1.

9. Rank Function for Vectors (F. F. Yao and A. C. Yao)

The problem of computing the rank function r and the longest chain c for a set of n d -component vectors is closely related to the problem of finding longest common sequences for alphabet strings. It is shown that r, c can be computed in $O(n \lg n)$ steps for $d=2$, $O(n(\lg n)^2)$ steps for $d=3$, and $O(dn^2 - \frac{1}{d-1})$ steps for $d < \frac{1}{2} \lg n$. It is also shown that, for $d \gg n^4$, r, c can

be determined in $O(dn^2/\log n)$ steps by employing a technique similar to the "Four-Russians method", used in fast Boolean multiplication. We also give an algorithm that takes $O(n^{1+\frac{1}{d}} (\log n)^{d-2})$ steps on the average.

10. Optimal Digital Search Trees (F. F. Yao and A. C. Yao)

Given N items with access frequencies $f_1, f_2, \dots, f_N$, it is desired to construct a digital search tree with minimal cost. We show that this problem can be reduced to a standard assignment problem with N persons and $N \log N$ jobs. The problem can then be solved by well-known methods.

11. Tiling with Incomparable Rectangles (A. C. Yao and E. M. Reingold)

Two rectangles are incomparable if neither can be placed inside the other when they are aligned so that corresponding sides are parallel. We derived results for the problem of tiling rectangular regions with sets of two or more mutually incomparable rectangles when all side lengths are integers. Among other results, it is shown that there are only a finite number of integer pairs N, M , $N \geq M \geq 10$, such that an $N \times M$ rectangle cannot be tiled with 7 incomparable rectangles.

Publication

F. P. Preparata and J. Nievergelt, Difference-Preserving Codes, IEEE Trans. Information Theory, Vol. IT-20, Number 5, pp. 643-649, Sept. 1974.

7. THEORY OF DIGITAL COMPUTER ARITHMETIC

(Supported in part by the National Science Foundation under Grant No. US NSF GJ 38204.)

7.1 A Study of Arithmetic Structure for LSI

The research in this quarter was centered mainly around the logical design of the arithmetic processor (multiplier-adder structure) of a Digit Processing Module (DPM). Major considerations in the logical design were two important LSI constraints:

- a) the need for as few different cells to construct the higher-radix arithmetic processor
- and b) the necessity for as few input and output pins as possible.

The logical design of the arithmetic processor section of a DPM is almost complete.

The necessity for few input/output pins has resulted in the discovery of a one-dimensional iteratively structured encoder to convert a signed-digit ($\bar{1}$, 0, 1) radix-2 number to an arithmetically equivalent number in sign-magnitude form.

Work is being continued on the development of a division algorithm suitable for an iteratively structured one-dimensional arithmetic unit of which the DPM forms one digital position.

(L. N. Goyal)

7.2 Continued Fraction Arithmetic

In this quarter, recent work on continued fraction arithmetic was summarized in form of a report entitled, "The Use of Riccati Equation in Digital Computer Arithmetic," UIUCDCS-R-74-674. It is shown that a very large class of functions can be evaluated by this

method. However, the selection procedures for the partial numerators and partial denominators are yet to be obtained. It is also pointed out that the main difficulty in the derivation of selection procedures is the unbounded growth of coefficients.

Towards the end of the quarter, it was realized that the growth of coefficients can be bounded by using appropriate digit sets for the partial numerators and partial denominators.

(K. Trivedi)

7.3 Function Evaluation Techniques

During this quarter work on a recently discovered efficient iterative method for evaluation of rational functions has been continued. The method is characterized by a simple, invariant algorithm which, when initialized with the set of rational function dependent coefficients as well as the argument, generates m most significant digits of the corresponding result in order of m additive steps. The range of coefficient and argument values is subject to certain restrictions, not incompatible with the usual restrictions of fixed-point arithmetic. A corresponding hardware implementation shows a great simplicity and modularity. Various properties of this method have been studied. A better proof of convergence as well as the rate of convergence has been obtained. Problems related to range extension and scaling are being investigated as well as a possible application of the method in mesh-type computations related to the partial differential equations and in solving tridiagonal systems of linear equations.

(M. D. Ercegovic)

7.4 Calculator Design

This quarter an M.S. thesis entitled "A Four Function Calculator" based on DeLugish's continued product theory was completed and is ready for printing. The machine on which the thesis is based was built and tested on the EXCEL equipment in the Logic Lab. of the Department. It worked according to theory when taking into account the limiting aspects of some of the hardware. Building the test machine revealed many of the problem areas to be encountered when designing and building an actual functional machine. The thesis will be published in the next quarter.

Research has now begun in the area of PLAs (Programmable Logic Arrays) and their applications in performing arithmetic operations. The PLA is a multiple output network in which the outputs are generated in a sum-of-products form. An upper array determines the intermediate product terms from the inputs given. These product terms are then fed into a lower array where they are summed (see Figure 1).

Figure 1

One can also perform sequential operations using PIAs. By adding a feedback loop from the output of the PIA into its inputs, the PIA becomes a sequential network. External feedback networks usually consisting of flip-flop elements are used to store the output for the next sequential step.

The current applications of the PIA include code conversion, table look-up, arithmetic logic, control logic, and character generation.

(Mary Jane Irwin)

8. ACSES, The Automated Computer Science Education
System on PLATO

(Supported in part by NSF under grant EC-41511)

Summary

Substantial progress has been achieved in the following areas:

- A. Lesson library: the FORTRAN sequence has been polished and a controlled semester-long experiment to test its effectiveness has begun in C.S. 103.
- B. Compilers: the COBOL compiler is complete, lexical and syntax analysis part of FORTRAN compiler are complete. The automatic error analysis package (syntax and run-time) has been added to the PL/1 compiler.
- C. Advising system: request processor and editors for the data bases: catalog, course outlines, student records, are complete.
- D. Automatic judging of programs written by students: the two test systems for sorting programs and for symbolic differentiation programs have started to run.
- E. KAIL, an extension of Tutor to include high-level control structures, is running.
- F. Automated exams: a project has been started.

More detailed descriptions follow.

J. Nievergelt

A. Lesson Library

During this quarter, much time was spent in planning for the PLATO experiment in C.S. 103 for the fall semester and in writing and finishing the FORTRAN lessons.

This fall semester, 25 students out of 50 enrolled in C.S. 103 are spending one hour a week on PLATO instead of in lecture. The PLATO hour is scheduled for the same time as the class meets, in order to keep the two groups separate. Also, the FORTRAN lessons are restricted so that students in the non-PLATO section will not be able to use them, even if they have access to PLATO through other courses. During the semester, the two groups will be compared on all tests, homeworks, and on attitude questionnaires.

In order to be ready for the fall, much lesson writing was done over the summer. Jeff Barber completed his work on an arithmetic lesson (fortarith). Greg Strass completed his IF lesson (fortif) and began a major overhaul of fortfmtl. Keith Slaughter finished his subroutine lesson (fortsubl). Ben Barta finished his lesson on do-loops (fortdo). George Friedman completed a major revision of fortarrayl and polished several additional lessons. Richard Montanelli got his introductory lesson (fortintro) operational.

To assist in debugging these and other FORTRAN lessons, ten C.S. 101 students enrolled in 1 hour of C.S. 199 during summer. They provided detailed critiques of the lessons which were quite useful to the authors.

Richard Montanelli

In September, Sandy Leach began an effort to finish and polish a number of lessons on which work had stopped short of completion. The first of these, numquad (numerical integration) was nearly finished.

George Friedman and Dave Eland added code to csrouter to give students the option of choosing a lesson through the guide or an index, or to directly enter the name of a desired lesson. (This last option is protected, so that the lesson named must be a computer science lesson.) Eland also added code to collect statistics on student usage of individual lessons.

H. G. Friedman

B. Compilers

The COBOL compiler and a reference manual for it (on PLATO) have been completed.

Bob Barnett

The lexical- and syntax-analysis part of the FORTRAN compiler are complete. Work has started on the run-time-package.

Mike Milner

This past 3-4 months, I have been implementing a prototype version of my automatic Error Analysis Scheme; this prototype system is installed in a developmental version of the general compiler system. At the present time, the error system works moderately well on the PL/1 language and tables. I am currently attempting to achieve this same working level with the

FORTRAN language and tables (and, perhaps, the COBOL language). When this is accomplished, I will then revise and refine the prototype model and attempt to obtain a stable, operational version of the compiler with the automatic error analysis system included.

Michael H. Tindall

The execution supervisor for PL/1 has been expanded to include character variables and their associated operators and built-in functions: concatenation, INDEX, SUBSTR, LENGTH, and VERIFY. The necessary changes were also made to the PL/1 syntax tables. The run-time package is now about 7300 words long which is dangerously close to the 8000 word limit on PLATO. As a result, it looks like there will be little expansion in the near future to the present PL/1 subset executed. Presently, the statement types accepted include PROCEDURE, DECLARE, assignment, IF THEN ELSE, GOTO, DO (all forms), GET LIST, PUT LIST, and END.

Ninety percent of the analysis of execution-time errors has been coded and debugged. Common Cause Analysis, which supplies the programmer with a list of possible causes for his error and discusses some of them with him, is complete. Value Analysis, which dynamically scans the students program in search of flaws which could have caused the execution error, is about 85% complete. Flaw Analysis, the simplest of the three analyses is only about 40% complete.

The error analysis routines were designed originally to be "use"d by each individual execution package. ("Use" is similar

to a macro call to an external macro which is compiled and embedded within the calling lesson's compiled code). However, due to PLATO's 8000 word limit, this method turned out to be unfeasible. The only solution that remained was to execute a jumpout from the execution supervisor to the error analysis routines which would reside in a different lesson altogether. The necessary redistribution of code to make this workable took quite a bit of time and accounted, in part, for the failure to complete the entire error analysis procedure by the end of this quarter.

Nevertheless, the development of the error analysis and written thesis are on schedule and should be completed by the end of the next quarter.

Al Davis

A study is being done to explore the effect of system interruptions on user performance at a terminal. Our compilers interrupt the student who is typing a program at the very moment he makes an error that can be detected by the compiler, and we would like to know whether or not this is a more effective way than the conventional approach, where all errors are signaled when the program has been entered completely.

A PLATO lesson administers the experiment, in which subjects must complete two simple tasks at the terminal. Half the subjects are interrupted immediately upon making an error, and must correct the error before continuing, while the other half are informed of errors only after they finish the task. Comprehensive data on subject performance are collected by the lesson, after which each subject

completes a standard personality test. Analysis will determine how performance is affected by method of error interruption and whether these effects depend on personality.

The pilot study has been completed and results show some statistically significant differences between the two error interruption method. The PLATO lesson is currently under revision in preparation for the full-scale experiment to be carried out in October.

Barr Segal

C. The Advising System: GUIDE

A working version of the GUIDE conversational information system is currently operational. A student can type in a request for information (an English sentence), and the system will respond by displaying the desired information (or an explanation will be given as to why the request could not be handled).

Recent past activity on the GUIDE system includes work in the following areas:

- (a) The translator has been rewritten in order to enable integration of the data base, clarify the flow of control, and reduce the amount of file space required.
- (b) Detailed design of the entire data base for the system has been completed. A data base editor has been implemented which facilitates the entry of information, maintains internal consistency, and manages the allocation of storage. Additional work on the data base includes the following:

- (1) lesson catalog: a record of all lessons (over 100) in the computer science PLATO library has been entered into the data base.
  - (2) course outlines: several course outlines have been entered.
  - (3) student record: routines to manage the collection and display of student data have been written
  - (4) concept space: a few keywords describing some of the lessons in the catalog have been entered.
- (c) A request processor has been implemented. The processor analyzes the intermediate representation of the original request, searches the proper data bases for the desired information, and generates a display which presents an appropriate response to the student.
- (d) Data collection on the operation of the GUIDE has been implemented. A record is made of every request presented to the system, along with a copy of the translation and response made by the GUIDE. This will facilitate analysis of the effectiveness of the system and help reveal areas that need improvement.

Current work on the system includes monitoring its performance under student use, development of a thorough description of each lesson in the library (with particular emphasis on the structure of relationships between lessons), and research into the content and structure of the concept space.

Dave Eland

D. Automatic Judging of Programs

Development of the system for interactively judging student-written PL/1 programs for performing symbolic differentiation has progressed to the operational prototype stage. A graph editor has been written in TUTOR to allow entry and editing of the AND-OR graph which the system employs as a representation of possible solutions to the problem. Other routines have been written to conduct an interactive dialogue with the student, traverse the graph in response to the students' input during his interaction, and comment on his approach to a solution. A preliminary vocabulary for use in the dialogue has been devised, and a small graph (about 20 nodes) spanning a logically complete portion of a solution to the general problem has been entered for use in testing.

Current efforts are concerned with refining the prototype's reaction to student inputs. Two approaches are being taken:

(1) protocol analysis of sessions on the symbolic differentiation problem between introductory level students and human tutors, as an aid to refining the solution graph and aligning the TUTOR vocabulary with that used by beginning programming students, and (2) closely supervised student use of the machine tutor, to improve the operational aspects of the student-tutor interface.

Ron Danielson

An editor and interpreter have been written for a small programming language for sorting, and for an assertion language in which assertions can be made about the state of the array to be sorted.

Prabhaker Mateti

E. KAIL, a High-Level Extension of TUTOR

The objective of the KAIL project is to propose an alternative to the TUTOR language in the PLATO environment. In designing KAIL, emphasis was placed on providing a good structure for the language since this is the area in which TUTOR is weakest. KAIL is an ALGOL-like language with procedures, block structure, and scope rules for variables, modifications, and user defined interrupts. It includes a powerful construct known as the selector which imposes a natural structure on the TUTOR judging mechanism. Various forms of the selector also serve the language as iteration statements, blocks, and the traditional selection statements, case and if-then-else.

A subset of KAIL was chosen for implementation and a compiler was designed, implemented, and debugged. The output of the compiler is TUTOR code which is condensed and run on the PLATO system.

David W. Embley
Fred Hansen

F. Automation of Exams

Two efforts have been started towards the goal of automating exams.

- 1) A table-driven generator of single statements (of specified type and complexity, in any of the programming languages implemented in the ACSES interactive programming system) is near completion. This program will be used together with a question generator and related grader, to produce a variety of test questions using single statements.

- 2) An interactive system for automatic examination of programming skills using the ACSES interactive programming system for checking and grading programs prepared by students. The PL/1 package is used initially; adaptation to other packages presents no difficulty. The existence of a problem-file is assumed. The student taking the exam types in a program as a solution and keeps interacting with the system, until either the system accepts the solution as correct or the student decides to quit. After completion of this interactive sequence, the student's score is computed according to data saved during the interaction. The score and other data of educational or administrative interest may be saved. The sequence of interactions and the grading criteria are partly specified by the problem author and partly build into the system. The instructor prescribing the examination has some control on the scoring criteria.

After completion of coding and debugging (planned for October 1974), extensive tests have to be carried out for testing the grading logic and in order to observe instructors' and students' reactions. Finally, the system will be useful for student's drills and for exams as well (on different problem files!).

A paper describing the system has been submitted for presentation at the 2nd IFIP World Conference, Computers and Education, Marseille, September 1975.

B. Barta
F. Izquierdo

9. MACHINE PERCEPTION AND INTELLIGENCE
(AQVAL Project)

1. Theory and implementation of variable-valued logic (Michalski)

First steps have been made toward development of a new VL system, called VL₂. One of the major new features of this system will be a new form of a selector, namely:

$$[f(x_i)_{i \in I} = R]$$

where $f(x_i)_{i \in I}$ is a propositional calculus or arithmetic function of variables x_i , $i \in I$, and R is a subset of the output domain of a VL₂ formula. The system includes also quantifiers. It can be briefly characterized as a 'variable-valued first order predicate logic'.

The concept of 'constant and variable decision spaces' have been developed and used as a basis for characterization of pattern recognition methods. Michalski has presented a paper 'A Variable Decision Space Approach for Implementing a Classification System' at the 2nd International Joint Conference on Pattern Recognition.¹

Work has been conducted (in cooperation with A. B. Baskin) on the comparison between grammatical inference and variable-valued logic research. Preliminary results on this topic obtained by Baskin have been described in the DCL report.²

Michalski was an invited speaker at the NATO Advanced Study Institute on the Computer-Oriented Learning Processes,

August 21 - September 6, in Bonas Francé (sponsored by the French Department of Defense and IRIA).

Presented by him was a paper entitled 'Learning by inductive inference'. This was an international meeting, only by invitation, devoted entirely to problems of machine learning. Represented countries: Greece, Britain, Japan, U.S.A., France, Belgium, Switzerland, Italy, Germany, Soviet Union, and Bulgaria. It was an extremely successful meeting in which all major directions of research in the field of machine learning were discussed and exposed to criticism and evaluation. The concept of simplicity as a guidance for learning and generalization used by us was generally approved (in a specially organized discussion on this topic). Variable-valued logic as a tool for decision making enjoyed a significant interest and approval among the participants.

2. Application of VL Systems (Conrad, Michalski)

The data on 20 liver diseases obtained from Dr. Croft has been reformatted and corrected. They will be used for the comparative study of the application of VL as a diagnostic and therapeutic decision making model. The data include more than 2000 cases of various liver diseases collected fro 15 years by Dr. Klatskin at Yale University. Yuen has debugged the major portion of the Uniclass Cover program.

Publications:

1. Michalski, R. S., "A Variable Decision Space Approach for Implementing a Classification System," Proceedings of the 2nd International Joint Conference on Pattern Recognition, Lyngby-Copenhagen, Denmark, August 13-15, 1974.
2. Michalski, R. S., "Learning by Inductive Inference," Proceedings of the NATO Advanced Study Institute on Computer Oriented Learning Processes, Bonas, France, August 26 - September 7, 1974.
3. Baskin, A. B., "A Comparative Discussion of Variable-Valued Logic and Grammatical Inference," Report No. 663, Department of Computer Science, University of Illinois, Urbana, Illinois, July 1974.

10. ITERATIVE SOLUTION OF SYMMETRIC AND NONSYMMETRIC
SYSTEMS WITH DYNAMIC ESTIMATION OF PARAMETERS

(Supported in part by the National Science Foundation under Grant Number
U.S. NSF-GF-36393.)

Several methods of solving large linear systems are being programmed
for numerical experiments. Two are for symmetric systems: the Conjugate
Gradient method and Diamond's Adaptive Tchebychev Iteration.

For nonsymmetric systems three methods are being implemented:

(i) The Conjugate Gradient method for the system $A^T A$; (ii) a modification
of the Bidiagonalization Method for solving the least squares problem
developed by Gene Golub, W. Kahan, et al.; (iii) an adaptive Tchebychev
iteration in the complex plane.

Method (iii) predicts complex eigenvalues from successive residuals
to improve iteration parameters. Because of the even-odd characteristic
of the Tchebychev polynomials, complex arithmetic may be avoided. Much
work has been done to find optimal iteration parameters.

Tom Manteuffel
Rick Roloff
Paul E. Saylor

11. MESH RESEARCH PROJECT

(Supported in part by the National Science Foundation under Grant No. US NSF GJ 36265.)

11.1 System Software

Three weeks were spent rewriting and preparing the final draft of a paper describing the asynchronous programming techniques used in our system. Two weeks were contributed to drafting proposals for continued network research. The remainder of the time was devoted to the preparation of our system for testing in a student-job production environment.

(G. Chesson and E. McClary)

11.2 Micro-programming Languages

In the last quarter a preliminary investigation of the design of a general emulation language (GEL) was made. This language will be designed to compile into the SUE micro-assembler language (SMALL). In recognition of the need for as optimal object code as possible, the GEL compiler will be designed to allow variable amounts of machine dependence in the language. The optimal amount of machine independence for efficiency in object code versus useful structure and ease of programming will be determined by the running of well understood benchmark programs, such as a routine for symbol table management for the PASCAL compiler written in microcode. Each of these benchmark programs will be written and rewritten in each of the generated dialects of the GEL language.

(S. Davidson)

11.3 Experimental Network Hardware

11.3.1 Maintenance

We experienced registration problems with the two clock cards (equivalent to DEC KW11-L) that were fabricated in house. The registration was improved and the trouble has not reoccurred. Registration on other cards made in house was examined and found to be marginal on some cards. These cards are either being removed from service or repaired.

Two standard memory boards were returned to the company for repair this quarter--one board has never worked and has been sent back twice before.

(P. Krabbe)

12. GENERAL INFORMATION

12.1 Personnel

The number of people associated with the Department in various capacities is given in the following table:

	<u>Full-time</u>	<u>Part-time</u>	<u>FTE</u>
Faculty	28	2	29.00
Visiting Faculty	5	0	5.00
Research Associates and Instructors	0	0	----
Graduate Research Assistants	0	62	30.83
Graduate Teaching Assistants	0	33	16.50
Hourly Junior Academic	0	27	9.28
Professional Personnel	1	0	1.00
Administrative and Clerical	15	0	15.00
Nonacademic Personnel (Monthly)	12	0	12.00
Miscellaneous Supporting Staff (Hourly)	0	16	5.03
TOTAL.....	61	140	123.64

The Department Advisory Committee consists of Professor J. N. Snyder, Head of Department, Professors M. F. Faiman, H. G. Friedman, C. W. Gear, D. B. Gillies, W. J. Hansen, G. R. Kampen, W. J. Kubitz, D. J. Kuck, D. H. Lawrie, J. W. Liu, C. L. Liu, R. S. Michalski, M. D. Mickunas, R. G. Montanelli, S. Muroga, T. A. Murrell, J. Nievergelt, D. A. Plaisted, W. J. Poppelbaum, S. R. Ray, E. M. Reingold, J. E. Robertson, A. H. Sameh, P. E. Saylor, D. L. Slotnick, D. S. Watanabe, and T. R. Wilcox.

12.2 Bibliography

During the third quarter, the following publications were issued by the laboratory:

Report Numbers

No.	639	Wong, Tze-Wah, "A System Monitor for Programs in Artificial Intelligence," September 1974.
No.	640	Goyal, Lakshmi N., "A Note on Atrubin's Real-Time Iterative Multiplier," July 1974.
No.	657	Stellhorn, William H., "An Experimental Information Retrieval System," July 1974.
No.	659	Weaver, Alfred C., "Data Compression for Character Strings," July 1974.
No.	660	Weaver, Alfred C., "On-Line Character Recognition," August 1974.
No.	663	Baskin, A. B., "A Comparative Discussion of Variable-Valued Logic and Grammatical Inference," July 1974.
No.	664	Varmelle, William and Lopez, Lawrence, "The PLW Compiler," July 1974.
No.	671	Michalski, R. S., "Learning by Inductive Inference," August 1974.
No.	674	Trivedi, Kishor S., "The Use of Riccati Equation in Digital Computer Arithmetic," August 1974.
No.	675	Yao, Andrew Chi-Chih, "On the Evaluation of Powers," August 1974.
No.	676	A Short Concordance to Laurence Sterne's <u>A Sentimental Journey Through France and Italy</u> by Mr. Yorick, Vol. I, A-L; Vol II, M-Z. September 1974 by Betty B. Pasta, David J. Pasta (Stanford U.), and John R. Pasta (National Science Foundation).

Theses

No.	636	Wen, Kuo Yen, "Machine Parameter Deduction by Program Analysis," August 1974. (M.S. Thesis)

Theses Cont'd

No.	662	Trivedi, Kishor S., "Prepaging and Applications to Structured Array Problems," July 1974. (Ph.D. Thesis)
No.	665	Kai, Joyce Moore, "A Text Editor Design," July, 1974. (M.S. Thesis)
No.	666	Young, Raymond P., "The Coder: A Program Module for Code Generation in High-Level Language Compilers," July 1974. (M.S. Thesis)
No.	668	Blandford, Dick K., "LINDA (<u>LINE</u> <u>D</u> rawing <u>A</u> nalyzer)," July 1974. (Ph.D. Thesis)
No.	669	Kopec, Stanley J., Jr., "Design Improvements in the COLFTAR Cooled Crystal Light-Value," July 1974. (M.S. Thesis)
No.	670	Roediger, Gary A., "Transient Viscous Flow Around an Elliptic Cylinder," July 1974. (M.S. Thesis)

File Number

No.	880	Gear, C. W., "Multirate Methods for Ordinary Differential Equations," September 1974.

12.3 Abstracts

Baskin, A. B., "A Comparative Discussion of Variable-Valued Logic and Grammatical Inference," Department of Computer Science Report No. 663, University of Illinois at Urbana-Champaign, July 1974.

Abstract:

This paper reviews what is meant by grammatical inference and it discusses some of the currently used methods of grammatical inference. The application of a variable-valued logic system to this inference problem is explored and, where possible, direct comparisons between methods are discussed. Examples of a correspondence between a variable-valued logic inference process and a grammatical inference process are presented. Operations which increase the class of languages to which variable-valued logic descriptions of classes of grammars are discussed. Type 3' and type 3" grammars are defined (both subsets of type 3 grammars) and the equivalence of variable-valued logic formulas and type 3' and type 3" grammars is illustrated.

Blandford, Dick K., "LINDA (LINe Drawing Analyzer)," Department of Computer Science Report No. 668, University of Illinois at Urbana-Champaign, Ph.D. Thesis, July 1974.

Abstract:

LINDA is a hardware project in pattern recognition. Line drawings in LINDA's vocabulary are made up of closed polygons and circles. Identification of such drawings is based upon the separation and serial analysis of each of the simple parts of the drawing. Line drawings are presented to LINDA on a clear plastic card, which a flying spot scanner converts to a video signal. This signal is analyzed in terms of the number of video pulses on a horizontal scan line. When the total number of pulses on one horizontal line changes, new regions are defined which may or may not belong to the same part of the line drawing. Long shift registers are used as delay lines and one horizontal line is delayed and compared to the next line. The result of this comparison gives the information needed to determine which regions belong to which parts of the line drawing. Once this information is obtained, the drawing is separated into its parts and each part is analyzed in turn with the top left most part being first. Twenty bits of information are produced for each part. These twenty bits tell how many corners the part has, how big the part is in relation to other parts, the location vertically and horizontally with respect to other parts, and whether the part has any connection to the outside of the line drawing. An associative memory is used for a dictionary with one card

and up to 160 bits for each line drawing. Information is received simultaneously by all cards in the dictionary as a series of 20 bit numbers. When the correct series of 20 bit numbers is received by a card in the dictionary it turns on a light indicating identification.

The LINDA system is a hardware project and it has been built in prototype form. Logical details of the construction are presented with a section on results and recommendations.

Gear, C. W., "Multirate Methods for Ordinary Differential Equations," Department of Computer Science File No. 880, University of Illinois at Urbana-Champaign, September 1974.

Abstract:

Multirate methods have often been considered for integrating systems in which one part changes much more rapidly than another; but, to my knowledge, little theoretical study has been made on the accuracy and stability of such methods. These few comments are intended to suggest some of the problems that might be involved, and some ways in which the methods can be analyzed.

Gold, D. E. and Kuck, D. J., "Implementation Considerations for Masking Rotational Latency by Dynamic Disk Allocation," International Workshop on Modelling and Evaluation Computer Architecture and Networks, Aug. 1974, IRIA, Paris.

Abstract:

Examples and implementation matters are discussed for an algorithm presented in another paper which masks rotational latency in memory hierarchies. The relations between I/O block size and random access memory cost are discussed. A number of questions about the implementation of the permutation algorithm are answered. Tradeoffs for system design or for use of these methods with existing systems are discussed.

Goyal, Lakshmi N., "A Note on Atrubin's Real-Time Iterative Multiplier," Department of Computer Science Report No. 640, University of Illinois at Urbana-Champaign, July 1974.

Abstract:

This correspondence presents a new multiplication algorithm for Atrubin's One-dimensional Real Time Iterative Multiplier such that all the cells, including the first cell in the array are identical as regards memory and control, for the no-delay case.

Kai, Joyce Moore, "A Text Editor Design," Department of Computer Science Report No. 665, University of Illinois at Urbana-Champaign, M. S. Thesis, July 1974.

Abstract:

A design for a text editor is developed and described so that it may easily be implemented without further development of the design. The editor avoids movement of text in memory by allowing the text to be stored non-contiguously. The editor provides commands to manipulate text and pointers to text, repeat sets of commands, create macros, and control the display of text.

Kopec, Stanley J., Jr., "Design Improvements in the COLFTAR Cooled Crystal Light-Value," Department of Computer Science Report No. 669, University of Illinois at Urbana-Champaign, M. S. Thesis, July 1974.

Abstract:

This report describes four improvements to the electronics of the COLFTAR (Cooled On-Line Fourier Transform and Reconstruction) System. These improvements are: the addition of more reliable crystal-temperature sensors, a new higher capacity write gun, synchronization of the entire system, and the addition of dynamic focusing to the write gun circuitry. These changes improve the reliability of the system and provide for improved resolution and contrast.

Kuck, D. and Muraoka, Y., "Bounds on the Parallel Evaluation of Arithmetic Expressions Using Associativity and Commutativity," Proceedings of the Seventh Annual Princeton Conference on Information Sciences and Systems, pp. 161-168, Mar. 1973; also appears in Acta Informatica, Vol. 3, Fasc. 3, pp. 203-216, 1974.

Abstract:

We show that any arithmetic expression with n operands and parentheses nested to depth d can be evaluated in at most $1 + 2d + \lceil \log_2 n \rceil$ steps, assuming that only associativity and commutativity are used to transform the expression. We also show that at most $\left\lceil \frac{n-2d}{2} \right\rceil$ processors are needed to achieve this bound.

Michalski, R. S., "Learning by Inductive Inference," Department of Computer Science Report No. 671, University of Illinois at Urbana-Champaign, August 1974.

Abstract:

The paper is addressed to learning processes which employ inductive inference. A system of variable-valued logic, called VL_2 , is briefly described and its application to implementing inductive learning processes is discussed. The VL_2 can be characterized as a 'multi-valued first order predicate logic'. An example of learning by a computer program the difference between two classes of objects is given.

Pasta, Betty B., Pasta, David J., and Pasta, John R., "A Short Concordance to Laurence Sterne's A Sentimental Journey Through France and Italy by Mr. Yorick," Vol. I, A-L; Vol II, M-Z, Department of Computer Science Report No. 676, University of Illinois at Urbana-Champaign, September 1974.

Abstract:

This short concordance to Laurence Sterne's A Sentimental Journey Through France and Italy by Mr. Yorick contains 26, 188 words of the 40,635 word text. Blocked words include certain articles, personal pronouns, parts of the verbs to be and to have, and the prepositions in, of, and to. The text was divided into logical episodes, and each word was tagged with the number of the episode in which it appears.

Preparata, F. P. and Nievergelt, J., "Difference-Preserving Codes," IEEE Trans. Information Theory, Vol. IT-20, Number 5; pp. 643-649, Sept. 1974.

Abstract:

A code (of integers by binary sequences) is called difference-preserving (DP-code) if it has the following two properties:

1. if the absolute value of the difference between two integers is less than or equal to a certain threshold, the Hamming distance of their codewords is equal to this value.
2. if the absolute value of the difference between two integers exceeds the threshold, then the Hamming distance of their codewords also exceeds this threshold.

Such codes (or slight modifications thereof) have also been called path-codes, circuit-codes, or snake-in-the-box codes. This paper discusses the application of DP-codes to pattern recognition and classification problems, and presents a construction of efficient DP-codes whose information content is asymptotically (in the length of codewords) of the order of theoretical upper bounds.

Roediger, Gary A., "Transient Viscous Flow Around an Elliptic Cylinder," Department of Computer Science Report No. 670, University of Illinois at Urbana-Champaign, M. S. Thesis, July 1974.

Abstract:

This paper presents a study of the transient flow of a compressible viscous gas around an elliptic cylinder which smoothly accelerates to a fixed velocity. The complete time-dependent Navier-Stokes equations are solved numerically in the body reference frame using a non-orthogonal time-dependent curvilinear coordinate system designed to describe the body exactly and to resolve the important features of the transient flow. The computation is performed in a fixed mesh which maps into a grid in physical space which evolves to accommodate the transient flow.

Sameh, A. H., Chen, S. C., and Kuck, D. J., "Parallel Direct Poisson and Biharmonic Solvers," Department of Computer Science Report No. 684, University of Illinois at Urbana-Champaign, July 1974.

Abstract:

We consider the direct solution of the Poisson and biharmonic equations using a number of arithmetic units in parallel. Assume an $n \times n$ grid of mesh points. We show that a finite difference approximation to the Poisson equation can be solved in $O(\log_2 n)$ time steps using $O(n^2)$ processors in parallel. This holds for Dirichlet, Neumann and periodic boundary conditions. We also show that a finite difference approximation to the biharmonic equation can be solved in $O(n)$ time steps using $O(n^3)$ processors in parallel.

Stellhorn, William H., "An Experimental Information Retrieval System," Department of Computer Science Report No. 657, University of Illinois at Urbana-Champaign, July 1974.

Abstract:

An experimental retrieval system designed to support data bases with little inherent structure is described. The initial data base contains the full text of several technical articles on information retrieval. Searching proceeds by means of a direct sequential scan through the data, and the user has very general control over the structure and context of the search. The system runs on a microprogrammable mini-computer, and several text searching and manipulation commands have been implemented in microcode.

Part I of the report is a user's guide. Part II contains detailed technical descriptions of algorithms employed for parsing, scheduling and executing a search request.

The system will be used to analyze strategies for efficient searching in this environment and to study the interaction between the system and a group of motivated users under controlled conditions.

Trivedi, Kishor S., "Prepaging and Applications to Structured Array Problems," Department of Computer Science Report No. 662, University of Illinois at Urbana-Champaign, Ph.D. Thesis, July 1974.

Abstract:

A demand prepaging algorithm is defined and proved to be an optimal demand prepaging algorithm. A variable-memory prepaging algorithm FWS is also defined and is based on Denning's WS algorithm. It is shown that FWS incurs zero page faults. Both of these algorithms, however, cannot be used in practice since

they require that the reference string be known in advance. Therefore, we describe several practical prepaging algorithms.

Data paging is of primary concern for problems with large data bases and for many types of array problems. In particular, we show that prepaging reduces the paging problems of array algorithms operating on large arrays. We also show that the use of a sub-matrix algorithm considerably improves the locality. Finally, we consider methods of automating these performance-improvement techniques by means of a compiler in the context of a structure-array-language.

Trivedi, Kishor S., "The Use of Riccati Equation in Digital Computer Arithmetic," Department of Computer Science Report No. 674, University of Illinois at Urbana-Champaign, August 1974.

Abstract:

This is a report of ongoing research in continued fraction arithmetic. It is shown that a large number of functions can be evaluated using the Riccati differential equation. However, the problem of properly selecting the partial numerators and the partial denominators remains to be solved. Several attempts to obtain such selection procedures are described. Possible directions for further research are indicated.

Vanmelle, William and Lopez, Lawrence, "The PLW Compiler," Department of Computer Science Report No. 664, University of Illinois at Urbana-Champaign, July 1974.

Abstract:

The PLW language and the structure of the first PLW compiler, done in assembly language and FORTRAN is detailed herein; eventually the compiler will be written in PLW itself. The PLW language is in many respects like PL/I. The object language of the compiler is FORTRAN. Hence the compiled code is, to a great extent, portable.

Weaver, Alfred C., "Data Compression for Character Strings," Department of Computer Science Report No. 659, University of Illinois at Urbana-Champaign, July 1974.

Abstract:

Two approaches to data compression in a telecommunications environment are examined: contiguous duplicate character compression and common phrase detection/replacement. Algorithms for each method are presented. Each method is shown to be useful for a given class of transmitted messages.

Weaver, Alfred C., "On-Line Character Recognition," Department of Computer Science Report No. 660, University of Illinois at Urbana-Champaign, August 1974.

Abstract:

This report discusses the current state-of-the-art in computer recognition of handwritten characters. Several current schemes are examined and criticized. A new recognition method is developed utilizing a voltage gradient tablet for input and clever software for essential feature extraction. A simulation program is included as an appendix.

Wen, Kuo Yen, "Machine Parameter Deduction by Program Analysis,"
Department of Computer Science Report No. 636, University of
Illinois at Urbana-Champaign, M. S. Thesis, August 1974.

Abstract:

In this paper, a generalized procedure is established to analyze Fortran programs in order to deduce some parameters of machine organization. The maximum possible average speedup and the corresponding efficiency for the set of Fortran programs are determined through the use of the Fortran Analyzer. Then a method of using operation type statistics to infer parts of machine structure is shown. In this paper, EISPACK programs are used. However, the same procedure can be applied to any set of meaningful Fortran programs.

Wong, Tze-Wah, "A System Monitor for Programs in Artificial Intelligence,"
Department of Computer Science Report No. 639, University of
Illinois at Urbana-Champaign, September 1974.

Abstract:

A lot of programs have been written dealing with Texture Recognition and other areas of Artificial Intelligence. The large amount of core these programs take up makes it impractical to have an entire program in core before execution. Therefore a MONITOR program that loads each sub-program at the time it is needed is desirable. Also the fact that the flow of control between these subprograms may not be easily predetermined makes it necessary that the MONITOR be flexible enough to allow programmers to easily modify the flow of control during execution. A MONITOR which fulfills the above needs in research in artificial intelligence was implemented on the IBM 360-75 under OS MVT release.20.6 at the University of Illinois Urbana-Champaign. In addition to general design principles of the MONITOR, the following paper will also give discussions of methods of setting up a supervisor-slave type system under the MONITOR, as well as possible ways of interfacing interactive console support to the MONITOR.

Yao, Andrew Chi-Chih, "On the Evaluation of Powers," Department of Computer
Science Report No. 675, University of Illinois at Urbana-
Champaign, August 1974.

Abstract:

It is shown that for any set of positive integers $(n_1, n_2, \dots, n_p)$, there exists a procedure which computes $\{x^{n_1}, x^{n_2}, \dots, x^{n_p}\}$ for any input x in less than $\lg N + c \sum_{i=1}^p [\lg n_i / \lg \lg (n_i + 2)]$ multiplications for some constant c , where $N = \max_i \{n_i\}$. This gives a solution to an open problem in Knuth [3] and generalizes Brauer's theorem on addition chains.

12.4 Colloquia

"Domain-Independent Automatic Programming," by Mr. Robert Balzer, Information Sciences Institute, Marina del Rey, California, September 9, 1974.

"Backtrack Programming Techniques," by Professor E. M. Reingold, Department of Computer Science, University of Illinois at Urbana-Champaign, September 30, 1974.

12.5 Drafting

During the third quarter, a total of 277 drawings were processed by the general departmental drafting section:

<u>Formal Drawings</u>	
Large Drawings	48
Medium Drawings	148
Small Drawings	31
Layout Drawings	3
Report Drawings	10
Change Order Drawings	27
Miscellaneous Drawings	10
Completed Total Drawings	277

(M. Goebel)

12.6 Shop's Production

Job orders processed and completed during the third quarter of 1974 are as follows:

	<u>Navy Grant</u>	<u>NSF Grants</u>	<u>Other</u>
Electronics Shop	21	1	35
Chemical Shop	24	0	21
Photographic Shop	15	3	19
Layout Shop	9	0	9

(F. P. Serio)

BIBLIOGRAPHIC DATA SHEET	1. Report No. UTUCDCS-QPR-74-3	2.	3. Recipient's Accession No.
	4. Title and Subtitle QUARTERLY TECHNICAL PROGRESS REPORT		5. Report Date
7. Author(s)		6.	
9. Performing Organization Name and Address Department of Computer Science University of Illinois at Urbana-Champaign Urbana, Illinois 61801		8. Performing Organization Rept. No.	
		10. Project/Task/Work Unit No.	
12. Sponsoring Organization Name and Address Department of Computer Science University of Illinois at Urbana-Champaign Urbana, Illinois 61801		11. Contract/Grant No.	
		13. Type of Report & Period Covered Quarterly Progress Report July - Aug. - Sept.	
15. Supplementary Notes		14.	
16. Abstracts Not applicable			
17. Key Words and Document Analysis. 17a. Descriptors Not applicable			
17b. Identifiers/Open-Ended Terms Not applicable			
17c. COSATI Field/Group			
18. Availability Statement RELEASE UNLIMITED	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 184	
	20. Security Class (This Page) UNCLASSIFIED	22. Price	

INSTRUCTIONS FOR COMPLETING FORM NTIS-35 (10-70) (Bibliographic Data Sheet based on COSATI Guidelines to Format Standards for Scientific and Technical Reports Prepared by or for the Federal Government, PB-180 600).

1. **Report Number.** Each report shall carry a unique alphanumeric designation. Select one of the following types: (a) alphanumeric designation provided by the sponsoring agency, e.g., **FAA-RD-68-09**; or, if none has been assigned, (b) alphanumeric designation established by the performing organization e.g., **FASEB-NS-87**; or, if none has been established, (c) alphanumeric designation derived from contract or grant number, e.g., **PH-43-64-932-4**.
2. **Leave blank.**
3. **Recipient's Accession Number.** Reserved for use by each report recipient.
4. **Title and Subtitle.** Title should indicate clearly and briefly the subject coverage of the report, and be displayed prominently. Set subtitle, if used, in smaller type or otherwise subordinate it to main title. When a report is prepared in more than one volume, repeat the primary title, add volume number and include subtitle for the specific volume.
5. **Report Date.** Each report shall carry a date indicating at least month and year. Indicate the basis on which it was selected (e.g., date of issue, date of approval, date of preparation).
6. **Performing Organization Code.** Leave blank.
7. **Author(s).** Give name(s) in conventional order (e.g., John R. Doe, or J. Robert Doe). List author's affiliation if it differs from the performing organization.
8. **Performing Organization Report Number.** Insert if performing organization wishes to assign this number.
9. **Performing Organization Name and Address.** Give name, street, city, state, and zip code. List no more than two levels of an organizational hierarchy. Display the name of the organization exactly as it should appear in Government indexes such as **USGRDR-I**.
10. **Project/Task/Work Unit Number.** Use the project, task and work unit numbers under which the report was prepared.
11. **Contract/Grant Number.** Insert contract or grant number under which report was prepared.
12. **Sponsoring Agency Name and Address.** Include zip code.
13. **Type of Report and Period Covered.** Indicate interim, final, etc., and, if applicable, dates covered.
14. **Sponsoring Agency Code.** Leave blank.
15. **Supplementary Notes.** Enter information not included elsewhere but useful, such as: Prepared in cooperation with . . . Translation of . . . Presented at conference of . . . To be published in . . . Supersedes . . . Supplements . . .
16. **Abstract.** Include a brief (200 words or less) factual summary of the most significant information contained in the report. If the report contains a significant bibliography or literature survey, mention it here.
17. **Key Words and Document Analysis.** (a). **Descriptors.** Select from the Thesaurus of Engineering and Scientific Terms the proper authorized terms that identify the major concept of the research and are sufficiently specific and precise to be used as index entries for cataloging.
(b). **Identifiers and Open-Ended Terms.** Use identifiers for project names, code names, equipment designators, etc. Use open-ended terms written in descriptor form for those subjects for which no descriptor exists.
(c). **COSATI Field/Group.** Field and Group assignments are to be taken from the 1965 COSATI Subject Category List. Since the majority of documents are multidisciplinary in nature, the primary Field/Group assignment(s) will be the specific discipline, area of human endeavor, or type of physical object. The application(s) will be cross-referenced with secondary Field/Group assignments that will follow the primary posting(s).
18. **Distribution Statement.** Denote releasability to the public or limitation for reasons other than security for example "Release unlimited". Cite any availability to the public, with address and price.
- 19 & 20. **Security Classification.** Do not submit classified reports to the National Technical Information Service.
21. **Number of Pages.** Insert the total number of pages, including this one and unnumbered pages, but excluding distribution list, if any.
22. **Price.** Insert the price set by the National Technical Information Service or the Government Printing Office, if known.