QUARTERLY TECHNICAL PROGRESS REPORT

October, November, December 1973

UIUCDCS-QPR-73-4

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, 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 IS UNLIMITED

24

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.

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK.

TABLE OF CONTENTS

	Page
1. CIRCUIT RESEARCH.....	1
1.1 LINDA (Project No. 28).....	2
1.1.1 Summary and Introduction.....	2
1.1.2 Project Status.....	2
1.1.3 Future Work.....	4
1.2 CAECOTRON (Project No. 43).....	4
1.2.1 Summary.....	4
1.2.2 Project Status.....	4
1.2.3 Future Work.....	5
1.3 NORMAN (NORMALizing ANALyzer) (Project No. 50)...	5
1.4 Molecular Stochastics (Project No. 51).....	8
1.4.1 Project Summary.....	8
1.4.2 Project Status.....	8
1.4.3 Future Work.....	10
1.5 OPTOBUNDLE (Project No. 65).....	10
2. HARDWARE SYSTEMS RESEARCH.....	13
2.1 FROG (Project No. 36).....	14
2.1.1 Summary.....	14
2.1.2 The Project (P) Element.....	14
2.1.3 Future Work.....	16
2.2 THESPIAC (Project No. 47).....	16
2.3 COLFTAR (Project No. 12).....	17
2.3.1 Summary.....	17
2.4 Digital Processing for Flight Simulation (Project No. 52).....	17
2.4.1 Summary.....	17
2.4.2 Realization.....	17
2.4.3 Future of the System.....	18
2.5 Digital Radio Aids for a Flight Simulator (Project No. 52-1).....	18
2.6 CM (Correlation Modulation) (Project No. 48).....	19
3. SOFTWARE SYSTEMS RESEARCH.....	21
3.1 Numerical Processes.....	21
3.1.1 DIFSUB with partial differential equations.....	21
3.1.2 New Stiff Methods.....	22
3.2 Graphical Support Programs.....	25
3.2.1 GPSS Flowchart to Source Translation...	25
3.2.2 Plot Package.....	31
3.3 Satellite Processor Software.....	37
3.3.1 OS/8.....	37
3.3.2 GRASS.....	37

	Page
3.3.3 ACID--Attached Controller for Interactive Displays.....	44
3.3.4 COMLO - GRASS: Remote Facility - PDP-10.....	53
3.3.5 Graphics Interpreter Language.....	57
4. SWITCHING THEORY AND LOGICAL DESIGN.....	79
5. MACHINE AND SOFTWARE ORGANIZATION STUDIES.....	82
5.1 Program Analysis.....	82
5.2 File Processing.....	84
6. COMPUTER SYSTEMS ANALYSIS.....	87
7. ASPECTS OF COMBINATORIAL COMPUTING.....	89
8. THEORY OF DIGITAL COMPUTER ARITHMETIC.....	91
8.1 Function Evaluation Techniques.....	91
8.2 Implementation of a High Speed Calculator.....	91
9. THE COMPUTER SCIENCE-PLATO PROJECT: REVIEW AND GUIDE- LINES.....	93
10. MACHINE PERCEPTION AND INTELLIGENCE.....	99
11. STATISTICS.....	101
12. MESH PROJECT FIRST ANNUAL REPORT.....	102
12.1 Introduction.....	103
12.2 Hardware and Software Support.....	105
12.2.1 Physical Configuration of the Developmental Facilities.....	106
12.2.2 Software Development.....	109
12.2.3 Firmware Debugging Facilities.....	116
12.2.4 Simulation Facilites.....	117
12.3 Current Research.....	118
12.3.1 Special Purpose Processors.....	119
12.3.2 Subnetwork Design.....	121
12.3.3 Language Design.....	124
12.3.4 Analysis and Evaluation of Allocation and Schedule Algorithms.....	124

	Page
13. GENERAL DEPARTMENT INFORMATION.....	128
13.1 Personnel.....	128
13.2 Bibliography.....	129
13.3 Abstracts.....	131
13.4 Colloquia.....	138
13.5 Drafting.....	139
13.6 Shop's Production.....	139

1. CIRCUIT RESEARCH

Summary

LINDA is nearing completion; Dick Blandford discusses improvement of the flying spot scanner and system tests. Bernard Tse describes preliminary work on using an adjustable delay between left and right image information to determine distance in the Caecotron project. The construction of NORMAN is under way, with a report on the A/D subsystem by Garlan Huberts. Jim Cutler has built a temperature-to-stochastic sequence transducer for the Molecular Stochastics project. There is also a report from Rob Budzinski on Optobundle, which is a new bundle-processing scheme that employs photons rather than electrons as the information carrier.

M. Faiman

1.1 LINDA (Project No. 28)

1.1.1 Summary and Introduction

LINDA (LINE Drawing Analyzer) is a hardware project in pattern recognition which seeks to identify two-dimensional line drawings. Each drawing is identified by separating it into simpler parts and identifying the parts serially. Information is extracted from each part to obtain its approximate shape, its size relative to other parts, and its position in relation to the rest of the drawing. Some judgment is then made as to what the whole line drawing is.

1.1.2 Project Status

During the previous quarters, work on LINDA has brought the system to the point where line drawings can be successfully separated into their simpler parts and each part can be identified. In the last quarter, problems with the flying spot scanner were overcome, the shading system was tested and was made operational, and the mechanical work of putting LINDA in a cabinet was completed.

An AGC system was built to correct the video amplitude shift caused by temperature problems in the flying spot scanner which could not be conveniently corrected. The circuitry is shown in Figure 1, and for the most part is self explanatory. The video amplifier output is sampled over several frames, and a voltage is produced proportional to its peak level. The photon-coupled isolator adjusts the gain of the amplifier so that the peak output voltage is about 4.5 volts when the peak input voltage varies from .5v to 1.5v.

The shading system discussed in the previous quarterly report was thoroughly tested and found to be adequate for this prototype system. No significant noise problems were encountered. Ground planes were used in

Figure 1. Automatic Gain Control on Video Amplifier.

critical parts of the system, and all lines over 10 inches long were run in twisted pairs or terminated co-ax.

Work on the dictionary logic is now underway, and the shading system is being used to provide test patterns as cards are completed. Nearly all the control logic is finished except for debugging. The whole dictionary system consists of the control logic and the dictionary proper, which has one card for each drawing in LINDA's vocabulary. The dictionary is an associative memory arranged so that new drawings may be easily added to the vocabulary or old ones removed conveniently. A complete discussion of the dictionary system will be given in the final report on LINDA.

1.1.3 Future Work

Work will continue on the dictionary system and it should be completed in the next few weeks. Some mechanical work remains to be done also.

Dick Blandford

1.2 CAECOTRON (Project No. 43)

1.2.1 Summary

In the Caecotron system, the pictures produced by two television cameras are quantized into two 16 x 16 matrix pictures. These are to be processed to extract depth information on the subjects in the original television pictures.

1.2.2 Project Status

A second approach to the determination of depth information from the two processed television pictures is currently under investigation. This approach, as described below, is simpler to implement than the algorithm described in previous reports; and for most circumstances it may work just as well.

In this approach, we limit ourselves to the global picture instead of focusing our attention on sections of it at a time. By comparing the right-hand picture with a delayed version (delayed by Δt_i) of the left-hand picture and by determining the value of Δt_i that would give a maximum degree of coincidence, we may say that the majority of the components of the television picture is at a distance that is inversely proportional to Δt_i from the camera.

1.2.3 Future Work

In the next quarter, the feasibility of this approach will be investigated.

B. Tse

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

The purpose of NORMAN is to recognize two-dimensional figures under conditions of rotation, magnification, and translation. The figure to be recognized is laid over a special array of phototransistors. Current is summed from this array by op amps on eighteen circuit boards, each of which represents a particular angular view of the figure to be recognized. During this latest quarter, the one hundred pin card edge connectors for these boards arrived and a card rack was built. Wire wrapping on the back of this rack is now being done.

The design of the analog to digital converter board was also finalized during this latest quarter. Figure 1 shows some of the details of this board. The three SN74161's form a twelve bit synchronous counter. The middle four bits of this counter are used to generate a staircase to which each one of the eleven analog input voltages is compared. These bits were chosen because the analog inputs are switched on when the count

Figure 1. A/D Board.

equals zero, and this gives enough time for the inputs to settle down before the staircase is generated.

The NAND gates at the outputs of the counter limit the staircase to twelve discrete steps corresponding to the numbers zero through eleven, since this is the range of numbers which are to be digitized. If the count were not limited in this way, the differential voltage limits at the inputs of the SN72510 comparators would be exceeded with the present gain configuration of the op amp.

The SN7407 is a high voltage open collector buffer which interfaces the TTL levels to twelve volts. The $\mu A715$ is a high speed op amp which does current summing of its inputs which are strung between positive and negative twelve volts in the same manner as the op amps which do the current summing from the phototransistors. This was done to minimize effects of power supply variations. If either supply varies it affects both the staircase and the analog inputs in the same way since the analog inputs are the outputs of the phototransistor op amps.

The SN72510 comparator's outputs go to the TTL level of "one" when the staircase is greater than the analog input voltage. This "one" is latched by the SN74111's which in turn shut off the strobe pulses to the "next" card. All of the counter's bits are fed to the "next" card and the center four are continually strobed into a latch until the strobe pulses are cut off. The last number strobed in is the correct digital number for that particular analog voltage. The latches will be on the "next" card because quite a bit of processing has to be done to these numbers once they are stored. The A/D card has to be built yet and the "next" card is now being designed.

Garlan Huberts

1.4 Molecular Stochastics (Project No. 51)

1.4.1 Project Summary

Molecular Stochastics is a project dealing with the measurement of physical quantities such as speed of a liquid in a pipe, temperature of a substance, etc. The uniqueness of this measurement is in the method of representation. Instead of obtaining a constant from the measurement, a stochastic sequence is produced. In short, a transducer is used for a measurement such as speed or temperature that will yield a stochastic sequence dependent upon this measurement.

1.4.2 Project Status

A crude transducer, shown in Figure 1, has been built for measuring temperatures. The temperature sensing device, which is simply a resistor, is a noise source in which the amplitude of the noise is dependent upon the temperature. Since the amplitude of this noise is very small (in the microvolt range), it must be amplified. The stochastic sequence is then obtained by buffering the amplified signal (to make it compatible with TTL) and putting it into a one-shot.

Several observations have been made using this transducer. The measurable temperature range may be varied by changing the gain of the amplifier or by changing the bias of the buffer transistor. Also, the resolution of this transducer may be improved by using a thermistor as the temperature sensing device, but the range is decreased. (Note that the thermistor is used as a noise source and not as a variable resistor dependent upon the temperature). The amount of the increase of resolution and of the decrease in range is unavailable at this time due to insufficient data.

Figure 1. Circuit Diagram of a Temperature Transducer.

1.4.3 Future Work

An oven has been located and further tests will be done using an improved transducer. As seen in Figure 1, a considerable amount of circuitry is needed to implement the original transducer. The improved transducer will eliminate some of this circuitry.

The effects of other noise sources in the transducer will be investigated. The end result is to minimize their adverse effects upon the stochastic sequence.

Work will start on another type of transducer which will measure velocity of a liquid inside a pipe.

J. Cutler

1.5 OPTOBUNDLE (Project No. 65)

OPTOBUNDLE is a new project using light fibers in a stochastic bundle processing machine. Previous bundle machines, such as BUM or SABUMA, have used wires in their bundles. At present OPTOBUNDLE is in the study phase; therefore a description of current ideas will be presented.

A bundle machine represents numbers in some finite range, say between 0 and 1 or between -1 and 1, by energizing part of the components in a bundle. For example, if one wanted to represent .5 on a 0 to 1 bundle, half of the components in the bundle would be energized. In OPTOBUNDLE, the components of the bundle will be light fibers. Light fibers provide several advantages over electrical conductors. First, as insulators, they can be used between two subsystems with a high potential difference. Secondly, light fibers are immune to electro-magnetic interference. Thirdly, fibers clad with an opaque material have very little

or no influence on adjacent fibers, that is, there is little or no cross-talk. Thus, light fibers can be used to carry information through hostile environments that ordinary conducting or propagating signals could not traverse.

OPTOBUNDLE will basically be a bundle arithmetic unit. The input to the bundle could be a light source controlled by a physical parameter. For example, a pivoted mirror with its angle of incident light controlled by ambient temperature could determine the portion of the optical bundle that is illuminated. Another possibility is a digital input. The digital to light bundle conversion could be accomplished by having one LED for each fiber. A digital input would then determine the number of LED's illuminated.

There are two basic approaches to implementing the arithmetic functions performed by OPTOBUNDLE. One approach is to randomly combine optical bundles in an appropriate manner to perform the operations. The other approach is to combine the bundles in a known manner. The fundamental element of multiplication, which is common to both approaches, is a negative logic passive AND gate, made by taking a fiber from each bundle and butting them against a single fiber which would be the output of the gate. If the input fibers are all dark, the output will be dark. In using negative logic, one assigns the value 0 to a bundle which has all its fibers illuminated, and 1 to a bundle with all its fibers dark.

When using the random approach, one combines the bundles randomly using the "AND gate" to implement multiplication. If we have a two input machine, calling one input value A, the other B, it can be seen that the number of output fibers will on the average be $A \cdot B$.

Addition is performed by combining the bundles to be added into one bundle. For example, if two bundles with 100 fibers each are to be

added, the sum bundle would contain 200 fibers. If one wanted to keep the number of fibers in a bundle at 100, one could take just 50 fibers at random from each bundle to form the sum bundle. The output of the sum bundle would be effectively multiplied by $\frac{1}{2}$.

The other method of implementing multiplication is by combining the bundles in a known manner as follows: Assume there are two bundles each with 100 fibers numbered 1 to 100. Then each fiber from one bundle would be combined with its correspondingly numbered fiber from the other to form an "AND gate." When representing numbers, one bundle would have the dark fibers adjacent to each other starting at 1. For example, if the number is .6, then fibers 1 through 60 would be dark. The other bundle would represent a number by distributing the dark fibers proportionally to groups of 10 fibers. For example, if the number is .4, the first 4 fibers of each group of ten would be dark, i.e. fibers X1 to X4 with X = 0, 1, ... or 9 would be dark.

If the bundles are implemented as described, it can readily be seen that the first 6 groups of 4 or 24 fibers will be dark in the above example. This method produces exact results only for inputs which are multiples of tenths. If one combines the fibers appropriately, other inputs could produce results which are rounded to the nearest hundredth.

The output could be detected by a photodetector at the end of each output fiber. The number of photodetectors that receive no light could be determined by some digital or analog technique, which could also display the result.

A practical method for subtraction and division have yet to be developed. Also, further progress is needed in the input and output sections of the project.

Robert Budzinski

2. HARDWARE SYSTEMS RESEARCH

Summary

Dev Bose describes the design of a P-(product) element that will be used extensively in the FROG project. Thespiac's digital crossfader was satisfactorily tested at the Krannert Center, and Les Daley has made some small additions as a result. Stan Kopec's report deals with a new write gun and thermistors for COLFTAR. Lucien Facchin has completed the design of the code keyer for the digital flight simulation project. In a related area, Gilles Garcia discusses the Cordic technique for rapidly computing flight parameters. Finally, Richard Kowall reports on how the CM reference signal is generated.

M. Faiman

2.1 FROG (Project No. 36)

2.1.1 Summary

FROG is a device which learns to discriminate between various classes of stimuli which for FROG are some bugs (both good and bad tasting) and predators. For details refer to previous quarterly reports.

The system is designed in terms of a set of specially defined logic elements. The hardware realizations of some of the logic elements were discussed in the previous quarterly reports. Realization of the remaining logic elements was considered in the last quarter. The following section gives a description of a product (P) element.

2.1.2 The Product (P) Element

If x_1 and x_2 are the inputs to a P-element, the output y of the element is the product, $x_1 x_2$, of the inputs.

The product of two 4-bit binary numbers x_1 and x_2 is easily obtained by a stochastic method. Figure 1 shows the block diagram of the P-element. The principle of operation is detailed below. Multiplication of two binary numbers by stochastic methods has been discussed in more detail by Y. K. Wo in his doctoral thesis (DCS Report No. 556).

The two binary numbers are compared in the two 4-bit magnitude comparators with two 4-bit random numbers that are derived from an 8-bit random number generator. For the generation of random numbers a pseudo-random 8-bit shift register sequence is used. As the generator is cycled through its states the comparator outputs produce two stochastic pulse sequences corresponding to the two numbers. The produce of the two numbers is formed by ANDing the two sequences. The output pulses from the AND gate are accumulated in an 8-bit counter. If the random number generator is cycled through a suitable number of states the accumulated number

Figure 1. Realization of a P-element.

(l.s.b. = least significant bit,
m.s.b. = most significant bit)

in the counter is the 8-bit product Y of the two 4-bit numbers X_1 and X_2 . The counter needs to be set to all zeros initially.

2.1.3 Future Work

Hardware design of the T*-element, the principal macro-element of the World Model, will be considered next.

D. Bose

2.2 THESPIAC (Project No. 47)

The THESPIAC project involves developing a digital memory system to control theatre lighting. During this quarter, progress has been made on two major aspects of the project: funding and design.

A proposal has been completed and is being submitted jointly by the Department of Computer Science and the Krannert Center for the Performing Arts to various organizations which may have an interest in supporting THESPIAC.

Parts necessary for the completion of a prototype digital cross-fader were received in November. Debugging of the prototype was then completed and the digital crossfader was tested in the Festival Theatre of the Krannert Center. The test was witnessed by Director Brotman, Lighting Director Caton, and Audio Director Spek. The performance of the fader was considered by them to be equal to that of analog crossfaders.

After the trial had proved that digital circuits will perform adequately, plans were made to expand the prototype to include drivers for at least one more dimmer, a second master fader, and a partial control unit. To make this possible, a small amount of money was made available by the Krannert Center to buy the required parts. A portion of this quarter was then spent expanding the tentative design plans to a level of

detail that allowed a parts list to be made. Purchase orders for these parts were then written and forwarded to the purchasing office.

Les Daley

2.3 COLFTAR (Project No. 12)

2.3.1 Summary

Due to delays in parts shipments, progress on COLFTAR slowed during the last quarter. Delivery of the new COLFTAR write-gun was taken early in the period, but its installation was postponed until the crystal-temperature sensing elements were received (just before the end of the quarter). Now, installation of the gun and the thermistor units can be performed at the same time, thus bringing the system up to atmosphere only once, rather than twice. It is hoped to have the system operational by mid-quarter. Design of the dynamic-focus system has begun, and should be finalized soon.

Stan Kopec

2.4 Digital Processing for Flight Simulation (Project No. 52)

2.4.1 Summary

One feature of the digital radio aids is the Code Keyer whose purpose is to produce the identification Morse Code for the station to which the radio is tuned.

In the last report (July to September, 1973) the principle of the Code Keyer was explained and a method was proposed and described.

2.4.2 Realization

During the Fall semester, the system was realized, tested and completed.

2.4.3 Future of the System

Due to a modification in the organization of the entire flight simulator, the Code Keyer system will be slightly changed, but will remain the same in its principle.

Lucien I. Facchin

2.5 Digital Radio Aids for a Flight Simulator (Project No. 52-1)

A small special purpose arithmetic unit has been studied using the "Cordic" computing technique. This technique was specially developed for use in real time digital computers where the majority of the computations involved discontinuous, programmed solutions of the trigonometric relationships found in navigational equations and required high resolution.

The Cordic AU consists primarily of three registers, three adder-subtractors, two shifting devices (barrel shifters) and special interconnections. A small read-only memory is also used to store constants. The control is stored in a RAM to allow control modification during the exploratory phase where this unit would be duplicated in different parts of a simulator.

Using the same basic algorithm the AU can compute elementary functions including multiplication, division, sin, cos, tan, arctan, ln, exp, and square root. The only operations required are shifting, adding and the recall of prestored constants.

The number of packages has to be minimal, and the unit has to be modular, in the sense that the precision can be chosen to be of 8, 16, 24 ... bits. The only problem comes from the barrel shifters which make these algorithms fast. It is difficult to build them as modular devices. So we decided to have them on separate cards. They are built out of the 58243

scalers from Signetics. The delay for shifting n places is 30 ns and does not depend on the number n .

A small read only memory is used to provide correction for the propagation of the sign bit (access time 33 ns).

For flight simulation 16 bits fixed point is enough and the equations are prescaled.

An 8 bit unit has been built and microprogrammed for solving angle and range in a triangle. The clock ran at 1.2 MHz, limited only by the fact that we have used a slow ROM for the constants. The time to solve this problem was 8.5 μ s. A 16 bit unit is now being studied with a bigger set of microinstructions.

Gilles Garcia

2.6 CM (Correlation Modulation) (Project No. 48)

Correlation Modulation is the encoding of information in the time delay between a reference signal and a delayed version of this reference signal. The amount of this delay is dependent on the amplitude of an analog signal which is the information we wish to communicate. As this analog signal increases, the time delay between the reference and the delayed reference increases.

The reference signal is a pseudorandom pulse sequence generated by the recirculating shift register technique. Feedback is taken from the outputs of the shift register in such a way that the particular set of outputs is different for each clock pulse for a maximum length of time.

For Correlation Modulation we will use only the last bit of the shift register in each clock period. It can be shown that any set of 8 consecutive bits of this sequence of 0's and 1's is different from any

other set of 8 consecutive bits, as long as we only look at a total group of 64 of these bits. As we will see in the near future, this is all we need for Correlation Modulation to work.

Richard J. Kowall

3. SOFTWARE SYSTEMS RESEARCH

3.1 Numerical Processes

3.1.1 DIFSUB with partial differential equations (M. Ostrar)

The use of DIFSUB to solve partial differential equations in conjunction with the technique of time stretching was continued. A list of stretching functions with which to experiment on Burgers' equation: $u_t + uu_x = ku_{xx}$ was drawn up for the initial and boundary conditions $u(x, 0) = \sin \pi x$, $u(0, t) = u(1, t) = 0$. It was decided to experiment at first with only one of these functions in order to see what general difficulties could be expected. The initial function chosen was $w(x, t) = \tanh^{-1} \left(x \tanh \frac{a}{2k(t+b)} \right) \frac{2k(t+b)}{a}$ where a and b could be adjusted as desired. Several unexpected problems have appeared in testing this stretching function, which is the reason that no results have been obtained. It appears that some of the simpler functions should be tried while the bugs for the above function are being dealt with. Testing and modification of ADAPT also continued. $u_t | w_x^2$ where $w(x, t)$ is the stretching function replaced u_x as the quantity which determines how ADAPT adjusts $w(x, t)$. The former was felt to be a better indicator of the x points where the largest errors might be introduced on a particular integration step. Since the scaled values of $u(x_i, t)$ are available from DIFSUB, and $w_x |_{x=x_i}$ is always available, no additional computations were introduced by this change. The restriction that $w(x, t)$ have four continuous derivatives with respect to x and two with respect to t was also dropped, as it was pointed out that unless w is specified for all x

and t , the distinction between smooth and discontinuous is irrelevant. Thus, w^{-1} no longer had to involve fourth degree polynomials. In fact, w_x became a sum of step functions.

ADAPT was tested on the equation $u_t = u + u_x + ku_{xx}$, $u(x, 0) = \sin \pi x$, $u(0, t) = u(1, t) = 0$. The limited results obtained thus far have been disappointing. Probable causes of the poor results are that ADAPT permits w_x to change too much between successive (time) integration steps and that centered instead of backward differences are used to compute u_x . If these shortcomings are corrected and no improvement results, the testing of ADAPT will probably be stopped.

3.1.2 New Stiff Methods (R. L. Brown)

The new numerical technique for solving stiff systems of ordinary differential equations using higher derivative multivalued formulas, first described in the previous quarterly report, has been partially implemented in a FORTRAN program D4 using techniques similar to those in DIFSUB. Among the features implemented thus far in D4 are an error estimator which allows the best next stepsize H to be chosen, and a corrector formulation which makes use of a Newton-like speed up of convergence in the corrector iteration.

The error estimator is dependent on the following theorem: For a convergent r^{th} order multivalued method based on evaluation of the 1^{st} through m^{th} derivatives to solve

$$y' = f(y, t), \quad y(t_0) = a \quad (1)$$

and for $r(y, t)$ continuous for bounded y , and for t in the range of integration,

$$e_n = y_n - y(t_n) = h^r \delta(t_n) + O(h^{r+1}) \quad (2)$$

where $\delta(t)$ satisfies

$$\delta'(t) = f_y(t) \delta(t) + \frac{C_{r+1}}{\Sigma \beta_i} y^{(r+1)}(t) \quad (3)$$

with $(t_0) = 0$, providing the starting values are exact. C_{r+1} is the error constant due to the operator $L_h(y(t))$ corresponding to the multivalued method, and $\Sigma \beta_i$ is the sum of the coefficients of $f_{n-i} = f(y_{n-i}, t_{n-i})$ in the formula.

This has been proven for $m = 2$ and the proof easily extends for $m > 2$. A theorem indicates that the local truncation error at each step is

$$\text{l.e.} = \frac{C_{r+1}}{\Sigma \beta_i} y^{(r+1)}(t) h^{r+1}, \quad (4)$$

which can be used to calculate the best next stepsize since $y^{(r+1)}(t) h^{r+1}$ is available from the dependent variable vector, stored in Nordsieck form, i.e. as the elements of a Taylor series.

The Newton-like speed up of convergence depends on the formula

$$F(\underline{y}) = 0 = \sum_{i=0}^k \alpha_{i,0} y_{n-i} + h \alpha_{0,1} f_n + \dots + h^m \alpha_{0,m} f_n^{(m-1)}, \quad (5)$$

$$f_n^{(i)} = \frac{d^i f}{d t^i} (y_n, t_n),$$

$$\alpha_{0,0} = -1.$$

Solving this for y_n as a Newton iteration gives

$$\begin{aligned} y_{n,(m+1)} &= y_{n,(m)} - \frac{dF}{dy_n} F(\underline{y}) \\ &= y_{n,(m)} + (I - h \alpha_{0,1} f_y - \dots - h^m \alpha_{0,m} f_y^{(m-1)})^{-1} F(\underline{y}). \end{aligned} \quad (6)$$

Since a subroutine DIFFUN (T, Y, Y', Y'',...) is available, numerical differencing can be used to calculate the matrix sum indicated. Either matrix inversion or Gaussian elimination and back substitution can be used to solve the iterated corrector.

The standard corrector for multivalue methods has been retained:

$$F(\underline{y}) = -\underline{y}_n + \Sigma + h \alpha_{0,1} f_n + \dots + h^m \alpha_{0,m} f_n^{(m-1)}, \quad (7)$$

$$\Sigma = \sum_{i=1}^k \alpha_{i,0} t_{n-i}.$$

Since the Nordsieck form is used to store the dependent variables, and the stepsize changes often, Σ is not directly obtainable by summation.

However, the predictor

$$P(t) = y_{n-1} + h y'_{n-1} + \frac{h^2}{2} y''_{n-1} + \dots + \frac{h^r}{r!} y^{(r)}_{n-1}, \quad (8)$$

$$h = t - t_{n-1},$$

represents the unique r^{th} order interpolating polynomial through the $r+1$ values $Y(t_{n-i})$, $i=1, \dots, r+1$, of the function satisfying

$$0 = \sum_{i=0}^k \alpha_{i,0} y(t - ih) + \alpha_{0,1} h y'(t) + \dots + \alpha_{0,m} h^m y^{(m)}(t), \quad (9)$$

$$k = r+1 - m.$$

Then $P(t_n)$ is an extrapolation of a polynomial which should satisfy within $O(h^{r+1})$ the equation

$$0 = \sum_{i=0}^k \alpha_{i,0} P(t_{n-i}) + \alpha_{0,1} h P'(t_n) + \dots + \alpha_{0,m} h^m P^{(m)}(t_n). \quad (10)$$

Thus

$$\Sigma = P(t_n) - h \alpha_{0,1} P'(t_n) - \dots - \alpha_{0,m} h^m P^{(m)}(t_n).$$

The major remaining difficulty in the program is in updating the elements of the Nordsieck vector in terms of the available information. Research in this area continues.

The availability of the higher derivatives for the subroutine is a significant problem. A system by A. C. Norman, called TAYLOR, features a compiler, written in BCPL, which takes as input a description of a system of differential equations using FORTRAN functions (SIN, TAN,...) and outputs a set of FORTRAN routines that will compute Y , Y' , Y'' , ..., enabling the problem to be solved by Taylor expansions. The system can currently be used to calculate the elements of the Taylor series at a point and make it available to D4. However, it is inefficient in the use of space and time, since it was developed for a different purpose. Possible changes to the program source are being considered to make it more efficient for its limited application. This will require the acquisition of a BCPL compiler.

3.2 Graphical Support Programs

3.2.1 GPSS Flowchart to Source Translation (R. Purvy)

The flowchart program has been modified considerably since the last description of the program's logic. The modifications have been partially to the program's function, but much more to the structure of the program, to make it more readable and, hopefully, more modifiable by persons other than myself. All but one of the things mentioned in the "Suggested Improvements" section of my thesis have been implemented. Three

external subroutines have been created by taking sections out of the main program, and, within the program modules, internal subroutines have been created. The code has been "cleaned up" by adhering more closely to 360 calling conventions. Finally, a great deal of documentation has been added.

The first functional change is to the method of parameter specification, proposed in "Suggested Improvements". Formerly, when the user added a block to his flowchart and wished to specify its label, operands, and comments, he had to make sure the parameters were all in order. When he hit the "modify parameters" function, something like this would come up:

LAB = DEFLT

BLOCK = ASSIGN

OPS

COM

OPS and COM were specified simply by extending with an equals sign and then the required information. LAB, or the label, on the other hand, has to be deleted first, and then retyped. For example, the user would delete the line and then insert 'LAB = LABEL' if he wished the block to have LABEL as a label. Furthermore, the program logic looked for an equal sign in each line, but did not check whatever was to the left of the equal sign; it assumed the parameters were in the correct order. The new system checks each line of text for one of the four keywords, LAB, BLOCK, OPS, and COM, and if it finds one followed by an '=', it puts the

text after the '=' in the appropriate column in the card image. Thus, the parameters can be in any order, and LAB will appear without the '=DEFLT' part it formerly had.

The second change is to the syntax analyzer for user commands, and is also described in my thesis. If the pictures of a flowchart are given names that end in numbers, e.g. PIC1, CH2, then this can be used to shorten the listing of the names in the COMPILE command. For example, if the flowchart was comprised of pictures PIC1, PIC2, PIC3, PIC4, CH1, CH2, CH3, and LAST, the user could type: COMPILER PIC1-4, CH1-3, LAST instead of: COMPILE PIC1, PIC2, PIC3, PIC4, CH1, CH2, CH3, LAST.

Furthermore, it is now possible to simply type in a GPSS program directly, without using the compilation facilities at all. By so doing, the user can take advantage of the tabbing and other options offered by the flowchart program, and get a card deck without even sitting down at a keypunch. To do this, he simply types 'ENTER' and then types in lines just as he would for the beginning and ending cards of a program produced through the compilation process. 'END' terminates the process, and the program will then ask if the user wants to run the program, or punch the deck, or both, or neither.

The node/connection tables, created by the flowchart program, give for each block in a picture all its connections. Each connection is represented by the terminal type involved and the instance number on the other end of the connection. For a picture with two blocks, connected to each other, the printout might be like this:

0	GENER		
		BOTTOM	1
1	TEST		
		TOP	0

the instance number and name appear on one line, followed by all connections for that instance.

This printout may be created by a "YES" answer to the question "PRINT NC TABLES?" which will appear in every run. The printout could be used for debugging a flowchart if error messages were produced, or if the source program derived from the flowcharts contained unexpected errors. It would be most useful in conjunction with a hard copy of each picture of the flowchart, using the hard copy device attached to one of the graphics tubes.

Two of the new external subroutines handle the first two modifications described here. One, PRMTRANS, analyzes the parameters, creates the parameter table for each picture, and translates all needed data in the subpicture block (produced by GRASS) to 360 format. The other, COMANL, analyzes the user commands and breaks down the COMPILE command into the separate pictures. The third new subroutine, MYPRT, is the routine whose address is placed in the output DCB of GPSS, and which is thereby entered whenever GPSS does a PUT to the print data set.

PRMTRANS creates the parameter table for each picture, described in the first XPR, 1973. It also translates sub-blocks one and three of the subpicture block into binary, in the case of numbers, and EBCDIC, in the case of characters. Sub-block two, the one containing the mnemonic parameters is left in its PDP-8 form. The keywords are present in ASCII in PRMTRANS for comparison purposes and each parameter is transferred to the card image in ASCII, and then the whole 80 byte record

is translated into EBCDIC. In the case of comments, precautions are taken so that, first, the comments will appear in column 36 or one space after the last operand, whichever is greater, and second, the comments will not extend beyond column 71. The latter is done to see if all the necessary lines of text are present; for example, the user could delete the 'BLOCK = ASSIGN' line, and the block field on the resulting card would then be blank. This imposes a certain responsibility on the user not to delete any necessary line of text; on the other hand, it allows deletion of unnecessary lines.

COMANL is the syntax analyzer for user commands. It must be passed, (as must all the external subroutines) in register one the address of a block of storage in the main program beginning with PTABADOR and ending with AIOATA. It returns its result in register 0, as follows:

- 0 - 'COMPILE'; no errors
- 1 - 'COMPILE'; syntax errors present
- 2 - 'ENTER
- 1 - 'COMPILE'; too many pictures to fit in PTAB

In the case of a COMPILE command, the picture names are placed in the area pointed to by PTABADOR. Each picture occupies 16 bytes, in PTAB, the first 8 being the picture name, and the other 8 being a pointer to the NC table for that picture, (described in the first QPR 1973) and a pointer to the parameter table. The latter is inserted by PRMTRANS and the former by the main program.

The syntax of the COMPILE command is as follows: 'COMPILE', one

blank, and the picture names, separated by commas. The command is considered to be terminated by a blank. In the case of something like 'PIC1 - 4', obviously there must be numbers on each side of the dash, and the second must be greater than the first. When a dash is found, all the numeric characters to the left make up the first number and all the numeric characters to the right of the dash up to the next comma or blank make up the second.

MYPRN is the routine that sends output from GPSS to the user terminal. Its functions are described in the thesis, so I will not go into it too deeply here. Each line sent from GPSS is saved in a block which is GETMAIN'ed by MYPRN, whose address is saved in BLADR. The number of lines present in this block at any given time is in NLINES. These two variables must be accessible to the main program (which has a 'EXTRN NLINES, BLADR' statement in it) for the following reason: when GPSS finishes and returns to the main program, there will generally be some text still not sent to the user terminal, (unless the last line sent happened to fill up the text block and cause it to be sent). Thus the main program has to do a CALL SEND and CALL REPLY. If the user replies 'P' the block is to be printed, and for that, NLINES and BLADR are needed.

The last subroutine has been external all along, C603. The restriction on GENERATE blocks described in the thesis, that they all must be in the first picture (in the order the pictures appear in the COMPILE command) has been eliminated in C603. Thus, the order of the pictures is now completely arbitrary, and affects only the physical form of the resultant source program, and not its logical form.

3.2.2 Plot Package (W. Chung)

A subroutine which can put up numerical labels with variable intervals for both x and y axes is now included in the plot package. The readability is improved by two factors: first of all, by these explicit numerical labels; secondly, by the fact that the max. of each axis is now the multiple of integers.

The subroutine, PAX, is written in assembler language and called once by the main plot routine.

Another subroutine that can draw a grid is written in FORTRAN and it is called once after PAX is called. The grid is constructed by this PGRID and saved in XFILE. The grid is displayed only when a user hits the command GRID.

The calling sequence of these subroutines are:

```
CALL PAX(ITEXT)
```

Input parameters:

```
ITEXT(1), ITEXT(2), ITEXT(3) = Xmin in A-format(E10.3)
ITEXT(5), ITEXT(6), ITEXT(7) = Xmax " " " "
ITEXT(9), ITEXT(10), ITEXT(11), = Ymin in A-format (E10.3)
ITEXT(13), ITEXT(14), ITEXT(15), = Ymax" " " "
ITEXT(4), ITEXT(8), ITEXT(12), ITEXT(16) = zero flag
```

```
if min, = -1
if max, = +1
if zero, = 0.
```

```
ITEXT(17) = Display origin for x-axis
ITEXT(18) = " " " " " " " "
ITEXT(19) = Length of axis in # of points on screen.
```

Output parameters:

```
ITEXT(1), ITEXT(5), ITEXT(9), ITEXT(13) = Fraction of
Xmin, Xmax, Ymin, Ymax.

ITEXT(2), ITEXT(6), ITEXT(10), ITEXT(14) = exponent of
Xmin, Xmax, Ymin, Ymax.
```

ITEXT(4), ITEXT(12) = starting position of labels for
x and y axes.

ITEXT(8), ITEXT(16) = interval size of labels for
x and y axes.

CALL PGRID (ITEXT, IXO, IYO, ISIZE):

ITEXT(4), ITEXT(12) = display position of the 1st label
for x, y axes

ITEXT(8), ITEXT(16) = size of display interval for x and
y axes.

IXO = display origin of x-axis.

IYO = display origin of y-axis.

ISIZE = display size of axis.

A typical output with a grid is shown below.

Feedback and message presentation is also improved: the plot package signals the acceptance of a hit by blinking the command which is hit, at the same time displaying a short message which guides a user to the next action sequence to be taken in a separate window, which does not interfere with the main display.

Knot Selection Algorithm

Following algorithms were tested for 5 examples with equidistant data points as input.

1) KNOT-B:

the slope at $X(5)$ is calculated by an interpolating cubic polynomial through $(X(J-3), Y(J-3))$, $(X(J-2), Y(J-2))$, $(X(J-1), Y(J-1))$, and $(X(J), Y(J))$.

2) KNOT-C

Both the slope and the ordinate at each knot is approximated by least-squares error approximation. Let m_j and s_j be the slope and the ordinate. Then the least-squares error is locally minimized

$$E_j = \sum_{i=1}^{n_j} y_{j_i} - S(x_{j_i})^2, \quad x_{j-1} < x_{j1} < x_{j2} < \dots < x_{j_{n_j}} < x_j$$

$$= F(m_j, S_j)$$

$$\frac{\partial E_j}{\partial m_j} = 0 \quad \text{and} \quad \frac{\partial E_j}{\partial S_j} = 0.$$

3) KNOT-D:

The slope at each knot is obtained by the least-squares error interpolation.

$$E_j = F(m_j)$$

$$\frac{\partial E_j}{\partial m_j} = 0$$

In each algorithm, we tried to maximize the mesh size until $||f(x) - S(x)|| > \text{EPS}$.

Tested examples are as follows:

$$F1: y = e^{-.5x} \sin 2x$$

$$F2: y = \sin x / x$$

$$F3: y = 1 - e^{-2x}$$

$$F5: y = 6x / (x^2 + 3)$$

$$F6: y = 1 / [1 + (x - 3)^2]$$

The results of experiments are show in the following tables.

ND = # of Knots

TEND = 10.0

NOT-B

Input: Continuous Input

(1) $EPS = 0.5 \times 10^{-3}$

	F1	F2	F3	F5	F6
ND	30	15	14	16	22
hmax	0.515	1.87	2.88	1.81	1.5
hmin	0.116	0.436	0.151	0.136	0.154

(2) $EPS = 0.5 \times 10^{-4}$

	F1	F2	F3	F5	F6
ND	50+m	27	28	28	38
hmax	0.35	0.63	1.2	1.2	1.1
hmin	0.102	0.225	0.08	0.08	0.084

(3) $EPS = 0.5 \times 10^{-2}$

	F1	F2	F3	F5	F6
ND	17	9	8	9	
hmax	1.22	1.96	5.	3.5	
hmin	0.282	0.75	0.432	0.573	

KNOT-D

Input = Equidistant Knots ($h_i = 0.05$)

(1) $EPS = 0.5 \times 10^{-3}$

	F1	F2	F3	F5	F6
ND	18	10	13	13	14
hmax	$1.4^{(h_{17})}$	$1.6^{(h_4)}$	$2.05^{(h_{12})}$	2.05	$1.75^{(h_{11})}$
hmin	$0.25^{(h_3)}$	$0.75^{(h_5)}$	0.3	0.35	$0.3^{(h_7)}$

(2) $EPS = 0.5 \times 10^{-4}$

	F1	F2	F3	F5	F6
ND	35	19	19	20	26
hmax	0.85	0.95	2.2	2.6	1.2
hmin	0.15	0.4	0.15	0.15	0.15

(3) $EPS = 0.5 \times 10^{-2}$

	F1	F2	F3	F5	F6
ND	10	5	9	7	9
hmax	1.95	4.45	2.9	3.75	2.15
hmin	0.45	1.6	0.55	0.6	0.75

KNOT-C

(1) $EPS = 0.5 \times 10^{-3}$

	F1	F2	F3	F5	F6
ND	19	15	18	17	19
hmax	0.92	1.44	0.72	1.14	1.7
hmin	0.02	0.02	0.38	0.08	0.02

(2) $EPS = 0.5 \times 10^{-2}$

	F1	F2	F3	F5	F6
ND	17		16	16	14
hmax	1.3		1.08	1.68	2.58
hmin	0.02		0.02	0.04	0.02

(3) $EPS = 0.5 \times 10^{-4}$

	F1	F2	F3	F5	F6
ND	27	19	20	21	24
hmax	1.1	1.32	0.72	0.74	0.8
hmin	0.2	0.06	0.22	0.24	0.2

3.3 Satellite Processor Software

3.3.1 OS/8 (C. Segre, R. Whyte)

A new option has been added to HELP. When the user types 'CTRL/U' a ' U' is echoed and the time block number of the current block may be changed by immediately typing the new number followed by a '*'.

The 'V' command in the symbolic editor has been enabled. This command lists on the inktronic printer in the same manner as the 'L' command lists on the teletype. (See OS/8 user's manual for more detail.)

Two new options have been added to SWAP. SWAP will now ask if the files on the current system device should be transferred to the disk. If a 'Y' is typed, the transfer is made. The second option allows five files at a time to be transferred to the disk (will not stop asking for files until 'ALT MODE' is typed) if the user types a 'Y' in answer to the question: 'APPEND FILES?'

3.3.2 GRASS (C. Segre, J. Stynes)

The core of the monitor for the new interpretive graphics system has been written. This core consists of the interrupt handler, the Scheduler, the Dispatchers for Input, and the I/O Interface to the teletype (operator's console). Flowcharts for all these are included in this report. A short description of each of these code segments follows:

Interrupt Handler

The interpretive system is run with interrupt on. All input from the IBM 360/75, the disk, the timer for each console, the PDP-10,

the disk, and the Operator's Console must also be acknowledged by the Interrupt Handler before any further processing may be done. Several flags used in the flowchart for the Interrupt Handler are explained below.

Scheduler

This routine is entered when the interpreted program does any output or a 'WAIT'. If output is being done ('OPFLAG' negative), no other console can be swapped in. Thus, the Scheduler loops on checking OPFLAG and trying to schedule output to Operator's Console. If no output is being done, the input-pending flags for the DPU, 360/75, timer, and PDP-10 are checked. If any of these is set, control goes to the Dispatcher for the particular device. If there is no input-pending, the program tries to schedule Operator's Console output and scans any input from the Operator's Console. After this, the program goes back to checking the input-pending flags. This loop is continued until an input-pending flag is set and a console is swapped in. Flags used in the Scheduler are explained below.

Input Dispatchers

Very simple routines (in theory) which swap in called-for console and start interpreted program at correct place.

I/O Interface to Operator's Console

Receives and outputs to Operator's Console. Received characters are saved in a buffer, when a carriage return is inputted, the operator input flag is set and all further input is flushed until flag is cleared.

The output routine prints from a buffer until a line-feed character is encountered. When this happens, any input echoing needed, is initiated. Flags used in flowchart are explained below.

'OPFLAG'--output status flag

- Ø: no output in progress
- 1: output to DPU in progress
- 2: output to DISK in progress
- 3: output to 360/75 in progress
- 1: output terminated with no error
- 2: output flushed by operator
- 3: 360/75 parity error on output

'INP360'--input pending flag for 360/75

- Ø: no input pending
- 1: input pending

'INPDPU'--input pending flag for DPU

- Ø: no input
- 1: input pending

'INPP10'--input pending flag for PDP-10

- Ø: no input
- 1: input pending

'TTMODE'--mode flag for Operator's Console output

- Ø: system message (console dependent and console independent) output
- 1: echoing Operator's Console input
- 2: user message output

INTERRUPT HANDLER

BY CARLO SEGRE

SCHEDULER

BY CARLO SEGRE

* NOTE - THIS IS A SUBROUTINE, SO CALLING FIELD MUST BE SAVED AND RESTORED BEFORE RETURNING.

DISPATCHERS FOR INPUT

BY CARLO SEGRE

-74-

I/O INTERFACE TO TELETYPE

BY CARLO SEGRE

'TTYBSY'--teletype busy flag

Ø: not busy

1: busy outputting

'INPOPR'--input pending from operator

Ø: no input

1: input present and not echoed yet

2: input present and not scanned yet

3: input present and neither echoed nor scanned

'TTSTAL'--'echoing caught up with input' flag

Ø: echoing not caught up

1: echoing caught up with input

3.3.3 ACID--Attached Controller for Interactive Displays (N. Hennegan)

ACID is the resident program in the PDP-8/I which handles up to 8 computek consoles. Basically the program has three operations. The first is to process and buffer display files sent from the monitor in the PDP-8 to any of the COMPUTEK consoles. The second is to buffer and maintain a text line at the bottom of the screen. Included in this operation are text editing facilities such as delete, change and insert. The third of these operations is to send text lines and joy-stick hits the monitor in the PDP-8.

Software Communication between ACID and GLASP

Communication between the two processors is accomplished by two software buffers in the PDP-8 (TFLAG8 and DFLAG8).

DFLAG8 This is the first word of one of these buffers
which is used for data transfers from the 8 to the
8/I (i.e. Monitor to ACID)

WORD 1

bit 0 - always set non zero when data transfer pending
bits 2-5 function
bits 6-8 bank in the PDP-8 which contains the data.
bits 9-11 console number

There are currently 9 functions for ACID to perform:

00(CENABL) Enable console (accept input from console)
01(CDSABL) Disable console
02(TXTIN) Input a text line into the refresh and text editing
buffer
03(DSPLAF) Input and process a file
04(DSPLAB) Input and process a block
05(DSPLAE) Input and process a entry
06(DSPLAT) Display a text line at some XY on screen
07(ERASE) Erase screen
10(SMON8I) Start MON8I

WORDS 2-50₈ These words have different uses depending upon the
function in word 1.

CENABL, CDSABL, ERASE, SMON8I--use only word 1.

TXTIN- WORD 2 Address of buffer in PDP-8
WORD 3 Character count

DSPLAB--WORD 2 Starting address of block to be displayed
 DSPLAE--WORD 2 Starting address of block which contains entry
 WORD 3 Starting address of entry to be displayed
 DSPLAT--WORD 2 Starting address of text buffer
 WORD 3 y coordinate of the location where the text
 line is to be displayed
 WORD 4 x coordinate of the location where the text
 line is to be displayed
 WORD 5 Character count.

TFLAG8 This is the start of the second buffer which is used for
 communication from ACID to the monitor in the PDP-8.

WORD 1

bit 0 always set when data transfer pending
 bits 4-8 interrupting character. The interrupting character
 is either a number from 1-20₈ representing the appropriate
 push button on the joy-stick box or the number 0,
 representing a carriage return.
 bits 9-11 the console number

WORDS 2-46₈ as with DFLAG have different interpretations. If
 the interrupting character is non zero, the coordinates of the cursor are
 determined and placed in WORDS 2 and 3 (WORD 2 contains the y coordinate).
 If the interrupting character is zero (a carriage return), WORD 2 contains
 the number of characters which are in the text line. WORD 3 is the start
 of the text line in 6-bit packed ascii.

General Program Description

All data transfer between the two CPU's is controlled by the 8/I (ACID) and as I have discussed all flags are software flags. When ACID is not processing a display file or outputting characters to the consoles, it is in a wait loop with the interrupt on. Every 200 ms. ACID checks the software flag DFLAG8 in the PDP-8 to see if any function is desired.

Console Vector- Each console has a 20 word octal area termed its console vector or CV.

CVECØ,	0	/ FLAG WORD
	0	/OUTPUT BUFFER NUMBER (BUFO)
	0	/OUTPUT ADDRESS POINTER (PNT0)
CDF Ø		/DATA BANK OF OUTPUT BUFFER (CDFO)
TAD I .-2		
ISZ .-3		
JMP EEXT		
	0	/INPUT BUFFER NUMBER (BUFI)
	0	/INPUT ADDRESS POINTER (PNTI)
CDF Ø		/DATA BANK OF INPUT BUFFER (CDFI)
DCA I .-2		
ISZ .-3		
JMP FEXT		
	0	/CURRENT POSITION OF TEXT CURSOR (PNTIN)
	0	/TIME OUT CLOCK (TXTCLK)
	0	/UNUSED

The first word of the console vector is a flag word which is used throughout ACID to indicate modes and states of each console. These flags are:

- DBIT currently processing a display
- TBIT currently in the text line refresh mode (the text line at the bottom of the screen is being constantly re-written. The console displays are of the storage type, and in order for changes in the text line to be clearly visible, it is necessary to continually re-write the text line.
- RBIT the display interrupt is requested to be turned on at next clock interrupt. (The display interrupt occurs when the display is ready to receive another character.) This is to allow either a refresh of the text line, or the start or restart of output to the console from a buffered display file.
- LBIT (long line bit) This indicates that the text line in the refresh buffer cannot be completely displayed in the span of one clock interrupt. Normally a text line is refreshed once every clock interrupt. At the conclusion of the refresh cycle, the display interrupt for the console is turned off and the RBIT is turned on. During each clock interrupt the consoles are polled

to see if the display interrupt is to be enabled.

When a display interrupt occurs, the interrupt handler tests to see if the console is currently doing a text line or display file and jumps to the appropriate routines.

GBIT communication with the PDP-8 is enabled

EBIT console is enabled (i.e. accept keyboard or display interrupt from this console)

UBIT underline pending bit. In the text line refresh mode an underscore is placed where the next character typed will be placed. (In the insert mode, the next character typed will be placed before the underscore position.) During the output of the text line, the output pointer is compared with the input pointer or text cursor. If equal, the UBIT is set. Upon receipt of next display interrupt, an underline character is sent to the display and the UBIT is cleared.

INSBIT currently in the insert mode. Any characters typed while in this mode will be placed immediately before the text line underscore.

Description of Words 2 - 17₈ of the Console Vectors

BUFO The second word of the CV (BUFO) contains the output buffer number. A bank and one half (6000 words) is divided up into 200₈ buffers. These buffers are dynamically allocated and freed and are used to store text lines and processed display files. The buffers

are linked in LNKLST, and BUFO contains the address in LNKLST of the current output buffer. The address in LNKLST contains either a zero which means that the input and output buffers are the same or it contains the address in LNKLST of the next buffer to be output. BUFAD computes the buffer address and data field from the buffer number.

PNT0

The next word in the CV is the output pointer. It contains the next address in the buffer which is to be output. When zero, the next word to be output is in the buffer linked to the present in LNKLST.

CDFO

This word contains the IOT to change the data field to the field of the output buffer. It is in the form 62n1 where n is the bank of the output buffer. The routine, which empties display files, transfers control to this address. The next three instructions

TAD I .-2

ISZ .-3

JMP EEXT

loads the accumulator with the next word to be output and increments the output pointer. Control then returns to the calling routine.

BUFI

This word contains the input buffer address in LNKLST and is used in the same manner as BUFO.

PNTI

This word contains the current input pointer (the address of the next available empty location). In the text line mode, this pointer points to the end

of the text line, and during refresh of the text line, the refresh cycle ends when the output pointer equals this pointer.

CDFI

This word and the following three words perform similarly to the CDFO routine except that the input buffer is filled with this routine.

PNTIN

This word contains the cursor or input pointer in the text line. (The character in this location is underlined.)

TXCLK

This word contains a value which is incremented at every clock interrupt when in the text line mode. The value is reset to approximately 60 seconds whenever a character is typed. Failure to type a new character within 60 seconds will cause a text line kill and exit from this mode. (This is to insure that no text line will be "burned" into the screen.)

Program flow when doing a display

Suppose that during a wait, ACID checks the DFLAG8 in the PDP-8 and realizes that a display file is to be input and processed for a particular console. Control then passes to DSPLAF with the interrupt off. The starting address of the file is read and placed in ADDR which will always contain the next address in the PDP-8 to be read. The console is tested to see if it is currently emptying a display file. If it is currently emptying a display file. If it is bit 0 of the last word in the buffer, the new display file is cleared and is tacked onto the end of this one. The last word of a display has bit 0 set to inform the output routines that it is the end of the display file and to stop the display.

The file is read word by word from the PDP-8, converted to COMPUTEK format, and buffered. Interrupt is enabled after the first channel read so that other consoles can be serviced while this input is continuing. If all buffers fill up, input is discontinued until a buffer is freed. In some instances buffer emptying may catch up to the input. When this happens, the display interrupt is turned off and the RBIT is turned on. At next clock interrupt, the display will be resumed.

Program flow when doing a text line

If a console is enabled and a character is pressed or a line is input from the PDP-8 into the text line buffer, the console enters the text line mode. A buffer is allocated using the same routines as used in displaying a file. PNTI, CDFI, PNT0, and CDFO are initialized and a few initial characters are placed in the buffer. The character or characters are loaded into the buffer with a padded blank at the right. The padded blank is necessary because the underline is displayed under the previous character and that if the input pointer is at the end of the line, a blank is necessary for underline to be where the nexted character typed will go. The TBIT and RBITS are set indicating that this console is doing a text line and that it is requesting the display to commence at next clock interrupt.

Upon receipt of another character, the character is placed at the location of the input pointer and if necessary a blank is added to the end of the line. When the text line contains more than about 20 characters, the refresh cycle spans more than one clock interrupt (1/60-th of a second). Cycle speeds of 1/30-th of a second become noticable to the eye so that rather than wait for the next clock interrupt to begin the refresh cycle

it is desirable to begin the next refresh cycle immediately. This is accomplished by turning the RBIT off and LBIT on. With the LBIT on, refresh begins immediately.

If a carriage return is encountered, the text line is packed two characters per word and loaded into the appropriate words of TFLAG8 in the PDP-8.

Program flow when encountering a joy-stick hit

When one of the buttons is pressed on the joy-stick box, the keyboard if enabled will interrupt. TFLAG8 is checked to see if the PDP-8 can accept input. If so the XY position of the cursor is read and loaded into the appropriate words in TFLAG8. The first word is set containing the button number and console.

3.3.4 COM10 - GRASS: Remote Facility - PDP-10 (B. Whyte)

With the completion of the PDP-10 interface work began this quarter on converting existing remote facilities on the IBM 360 to the PDP-10. Specifically this involves the subroutines contained in COMMUNE.

The Fortran calling sequences for the routines TEXT, LINE, and LINED remain unchanged. These sequences may be found in the Remote Facilities Guide report #466 on pages 7-8.

Work was also begun on construction of a stand alone program in the PDP-8 to replace React which communicates with the 360. Upon its completion users will be able to utilize the facilities of the PDP-10 for truly interactive graphics and simulation.

LD8I - OS/8 ACID Loading Program

Due to the existence of and further development of various versions

of the terminal controlling program (ACID) which is resident in the PDP but which is edited and assembled in the PDP8, a generalized loading program was implemented to allow these to be easily put to use and tested.

The heart of the program is the MON8/I program which is resident in the 8/I and controls communication with the PDP8. Around this software channel LD8I contains various calls to the OS/8 operating system of the PDP8 which performs the functions of keyboard communication, file lookup, and device I/O.

To load a version of ACID into the PDP8/I it is first necessary to load and run MON8I. The instructions for this are contained in the GRASS: Terminal User's Guide, Report #467 page 34. Next the ASCII file of ACID must be assembled and saved using the SAVE operation of OS/8. (refer to OS/8 User's Manual page 9-25). The user then runs the LD8I program and responds to it by typing the name of and device on which the file is located. The file will be loaded in the 8/I and the PDP8 will return to the command mode of OS/8.

A typical teletype page would resemble this (PSP8 responses are underlined)

```

. R LD8I
* optional device name: ACID. optional file extension
.
:
```

Note a) if no device is specified LD8I assumes the current system device
b) if no file extension is given LD8I assumes .SV

Revisions to GDUMP (GRASS System Memory Dump)

The output format of the dump has been changed to resemble

existing dump outputs in order to make it more familiar to the user. There are now eight columns of digits followed by a column of their ASCII character equivalents. The dump now looks like this:

```
35040 7240 2022 1707 5520 1103 3700 1107 1344: PROG-PICK @ I
35050 4000 1151 07160 0422 0127 3300 2405 3024 @I)GNDRAW
```

TLK10 - PDP-8 Software Interface to the PDP-8

The software interface running in the PDP-8 has been designed to simulate a typical remote teletype terminal, thereby making the PDP-8 invisible to the PDP-10.

Basically the program rests in a loop testing pertinent device flags, servicing any which are raised and then continues on. Following is a description of these flags and the actions performed. The devices serviced are in order:

PDP-10 input to 8

Teletype output

Keyboard Input - output to 10

If input is pending from the PDP-10 the character is read and deposited in the input buffer. Next if the teletype is currently busy, TLK10 returns to test for PDP-10 input. Otherwise the character is sent to the teletype. In the case where no input is coming from the PDP-10 keyboard input is checked and if it exists it is sent to the PDP-10. Finally the input buffer is tested for characters. If it is not empty TLK10 jumps to test the teletype flags otherwise control returns to the top of the loop.

Several additions to this simple scheme were found to be necessary or particularly convenient. These include monitoring input buffer overflow and generating appropriate responses, keyboard clearing of the buffer, and the ability to output to the line printer.

In its current state TLK10 begins by sending the message "SET TTY PAGE" to the PDP-10. This enables the X-OFF ($\uparrow$ S) and X-ON ($\uparrow$ Q) keys on the terminal which respectively suspend and restore typeout from the PDP-10. In the event of input buffer overflow the X-OFF character is automatically sent by the PDP-8. When the buffer is cleared the X-ON is sent.

Another addition was the detection of the $\uparrow$ C character from the keyboard which clears the PDP-10 output buffer. However it is also necessary to clear the PDP-8 input buffer in order that the command mimi that on a standard 10-terminal.

Finally the ability to use the line printer as a faster means of retrieving the PDP-10 output was implemented. This required echoing of the keyboard inputs on the teletype by TLK10 as all other key echoes come from the PDP-10 and the placement of line printer flag tests in the previously mentioned loop. To use the line printer raise the bit 11 switch of the switch register.

Future additions might include the use of other peripheral devices such as DECTape, Disk, and visual screens as simulated PDP-10 output devices.

3.3.5 Graphics Interpreter Language (J. Stynes)

This quarter a preliminary instruction set was drawn up for a graphic interpreter language. The instruction set is broken up into the following areas:

- I Arithmetic Commands
- II Control Commands
- III Text Buffer Commands
- IV Data Structure Commands
- V Input/Output Commands
- VI Pseudo-Ops and Special Commands

Before listing the instruction set, it is first necessary to explain the different types of instructions and operands.

The different instruction types are listed in Figure 0.

The various symbols used to describe operands follow:

- () operand may be omitted. If so, a default value is assumed.
- [] operand may be omitted. If so, operand is on the top level of the stack. If the operand specified in the instruction is an N type (see below) a PUSH instruction is generated before the current instruction, to push the operand onto the stack.

For example, if an instruction is described as INST [N], then INST N generates:

PUSH N

INST

If the operand specified in the instruction is an S type (see below), a MOVE instruction is generated to move the operand to the top of the stack. For example, if an instruction is described as INST [S], then INST [S] generates:

INSTRUCTION TYPES

Type

Format

1

1A

1B

used only for
MOVE instruction

2

3

5

used only for
INIT B instruction

Each word is 12 bits long

FIGURE 0

MOVE /-1,S

INST

S denotes that the operand refers to a level of the stack.

A stack level is denoted by preceding any number by /.

N denotes that the operand is either a number or a symbol equated to a number

A denotes that the operand is an address

T denotes that the operand is a text string of the form 'text string', or the address of a text string

If an operand contains more than one of the above letters, the operand may be any of the types denoted by the letters specified. For example

SN means that the operand may be either a stack level or a number.

I Arithmetic Commands

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
1	NEG		Negate the top word of the stack
1	COM		Complement (bitwise) the top word of the stack
1	OR	[SN]	Pop the top two levels of the stack, logically "or" them together, and place the result on top of the stack
1	AND	[SN]	Pop the top two levels of the stack, logically "and" them together, and place the result on top of the stack
1	ADD	[SN]	Pop the top two levels of the stack, take their arithmetic sum and place the result on top of the stack

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
1A	CSHFT	N	the top of the stack is circularly shifted N bit places (In a circular shift, the low order bits which are shifted off the end on the right are shifted back into the word as the high order bits on the left). If N is positive, the number is shifted to the right. If N is negative, it is shifted to the left
1A	ASHFT	N	The top of the stack is shifted right arithmetically. (In an arithmetic shift the bits shifted off the right end are ignored, and the high order bit is duplicated on the left.) If N is positive, the number is shifted right. If negative, it is shifted left.
1B	MOVE	S,SN	<p>If the second operand is an N, the number, N, is moved into the stack level specified by the first operand S. Note that MOVE S,N generates: PUSH N MOVE S,-1</p> <p>If the second operand, is an S, the contents of the level specified by the second operand are moved to the level specified by the first operand. The contents of the level specified by the first operand and all other levels are</p>

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
-------------	-----------------	----------------	--------------------

adjusted accordingly. Note that the symbol /-1 has special significance. (See examples below)

Some examples follow:

- a) MOVE /2,/7: The contents of the seventh level are moved to the second level. The second level and all those below it are moved down one level.
- b) MOVE /-1,/3: The third level is pushed onto the top of the stack
- c) MOVE /4,/ -1: The top of the stack is popped and its contents are stored in the fourth level

1	PUSH	N	N is pushed onto the stack. Note that a PUSH command is denoted by a 0 in the high order bit. The rest of the word is used to represent the number N. Note also that rather than PUSH N, just N may be written.
1A	POP	SN	The top N levels of the stack are popped. If operand is S, the stack is popped down to the level which was equated to the operand. (See Pseudo-Ops, the description of =)

II Control Commands

In the following branch instructions, if P is appended to a branch instruction, the top word on the stack is popped after the instruction is executed. (It is popped regardless of the success or failure of a conditional branch)

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
2	BZ	A	If the top of the stack is zero
2	BZP	A	branch to A.
2	BNZ	A	If the top of the stack is non-zero
2	BNZP	A	branch to A.
2	BM	A	If the top of the stack is less than
2	BMP	A	zero branch to A.
2	BP	A	If the top of the stack is greater
2	BPP	A	than zero branch to A.
2	BPZ	A	If the top of the stack is greater
2	BPZP	A	than or equal to zero branch to A.
2	BMZ	A	If the top of the stack is less than
2	BMZP	A	or equal to zero branch to A.
2	BR	A	Unconditionally branch to A.
2	BRP	A	

III Text Buffer Commands

Refer to Figure 1 for a description of the text buffer.

TEXT BUFFER

LENGTH contains the length (in characters) of the character string contained in the buffer

P1, P2 are pointers which contain the number of the character they are pointing to in the buffer. The character buffer is numbered from left to right starting with 1

(Note: $1 \leq P1, P2 \leq \text{LENGTH}$)

These pointers are set by various text buffer instructions.

FIGURE 1

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
	TXTNUL		The text buffer is blanked out. It sets $P1 = P2 = 1$, $LENGTH = 0$.
2	TXTCON	T	The given text string T is concatenated onto the text string in the buffer. P1 remains the same. $P2 = P2 + \text{Length of T}$ $LENGTH = LENGTH + \text{Length of T}$
1	TXTCNV		The top word of the stack is popped, and converted into a 6-bit ASCII number. It is then concatenated onto the string in the text buffer. P1, P2, and LENGTH are adjusted as in TXTCON
1A	TXTDEL	N	N characters are deleted from the text buffer starting with the character furthest to the right. P1 remains the same. $P2 = P2 - N$ $LENGTH = LENGTH - N$
3	TXTCMP	T,A	The text string, T, is compared with the text string between P1 and P2. If they are equal the program branches to A. If they are unequal, the next instruction is executed. P1, P2, LENGTH remain unchanged in either case.
1	TXTSTP		Store P1 and P2 from stack into text buffer header. (It is assumed that P1 is on the top level of the stack, and P2 is on the second level)

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
1	TXTLDP		Push P2 and P1 onto the stack. (After the operation, P1 is on the top level)
1	TXTLEN		Push the LENGTH onto the stack.

IV Data Structure Commands

The following discussion refers to Figures 2, 3, and 4.

There are three basic units in the data structure: files, blocks, and entries. A file simply contains a file header followed by any number of blocks. The file header is used almost entirely by internal routines, and only the current block pointer, BP, may be modified by the user.

A block contains a block header followed by any number of entries. The block header is accessible to the user, and is set when the block is created. It describes the type of information contained in each of its entries. A description of the block header follows: (See figure 3)

WORD 0: contains the pointer to the next block in this file. (May not be modified by user)

WORD 1: contains the block number

WORD 2: contains the number of the current entry being referenced. (This is called the current entry pointer, EP)

WORD 3: contains the number of the current line being referenced. (This is called the current line pointer, LP)

WORD 4: contains the number of information words in each entry (See description of entry below)

WORD 5: contains the number of sets of x, y quads in each entry (See below)

WORD 6: Dx - the offset to be added to each x coordinate of all x-y quads (See below)

WORD 7: Dy - the offset to be added to each y coordinate of all x-y quads. (See below)

Entries are the structures which contain the bulk of the user's information. They can contain any or all of the following types of data: (See figure 3)

- 1) Information words - these are words which are allocated in an entry, and are used to hold any information which can not easily be contained in any of the other types. The number of these words allocated per entry is contained in the block header.
- 2) X-Y, Quads. These are sets of words (4 words per set) which are used to contain x,y coordinates. Note that one set corresponds to the number of words needed to describe a single line. The number of these sets per entry is contained in the block header.
- 3) Text group - A text group consists of the following: (See figure 4)
 - WORDS 1,2: The y and x coordinates, respectively, of the leftmost character of the first line of text in this text group
 - WORD 3: The length of the first text line in this text group
 - WORDS 4,N: The first line in this text group (in 6 bit ASCII)
 - WORD N+1: The length of the second line in this text group.
 - WORD N, N+M: The second line in this text group. (If this entry were to be displayed, this line would appear directly under the first line)

WORD N+M+K: The last line in this text group

Note that the total number of lines in a text group may vary from entry to entry, but that all entries in a particular block have the same number of information words and sets of x, y quads. The commands for the data structure are divided into three major parts:

- 1) File Commands
- 2) Block Commands
- 3) Entry Commands

1) File Commands

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
1	SETFP	[SN]	The top level of the stack is popped and stored in the FP.
1	RDFP		The FP is pushed onto the stack.
1	INITF	[SN]	An active file is created. The top level of the stack is popped, and the new file is assigned this number. The FP is set to this new number.
1	FLUSH	[SN]	The top level of the stack is popped, and the active file associated with this number is flushed.

2) Block Commands

All Block commands refer to the block contained in the file pointed to by the FP. (This pointer is kept internally.)

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
1	SETBP	[SN]	The top level of the stack is popped and stored in the BP. The EP, LP are set to 1.

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
1	RDBP		The BP is pushed onto the stack
1	DELB		The block pointed to by the BP is deleted.
1	SETDX	[SN]	The top level of the stack is popped, and stored in the DX word of the block pointed to by the BP.
1	SETDY	[SN]	The top level of the stack is popped, and stored in the DY word of the block pointed to by the BP.
5	INITB	N1,N2,N3 (,DX,DY)	<p>A Block is created in the file. The BP is set to point to this block. The EP in this block is set to 0, LP is set to 1.</p> <p>The operands are as follows:</p> <p><u>N1</u> is the block number</p> <p><u>N2</u> is the number of words of interpreter information to be assigned to each entry in this block.</p> <p><u>N3</u> is the number of sets of x-y quads to be assigned to each entry in this block</p> <p><u>DX,DY</u> are the displacements to be added to x-y coordinates of x-y quads. (Default is DX = DY = 0)</p>

3) Entry Commands

All entry commands refer to the entry contained in the block pointed to by the BP, which is contained in the file pointed to by the FP.

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
1	SETEP	[SN]	The top level of the stack is popped and stored in the EP. The LP is set to 1.
1	RDEP		The EP is pushed onto the stack.
1	DELE		The entry pointed to by the EP is deleted.
1	APPE		An entry is created at the end of the list of entries for this block. The length of the entry is based on the number of information words allotted and the number of sets of x-y quads allotted. Minimum length of an entry is 1. (The displacement to the next entry is needed.) The EP is set to point to new entry, LP is set to 1. Note: Total number of words allotted is: 1 + number of information words + 4 (sets of x-y quads)
1	INSE		same as APPE except that the entry is inserted before the entry pointed to by the EP. The EP is then set to point to new entry.

The instructions which deal with the information inside the entry are broken up into two types

- 1) Information Word and x-y quad commands
- 2) Text group commands

All of the following commands deal with the entry pointed to by the current EP.

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
1A	LDW	N	The Nth word of the entry is pushed onto the stack. (The first information word is numbered 1)
1A	STW	N	The top level of the stack is popped and stored in the Nth word of the entry.
1A	LDYXW	N	The x-y coordinates in the N and N+1 words of the entry are pushed onto the stack (After the instruction has been executed the Nth word is on the top level of the stack)
1A	STYXW	N	The top two levels of the stack are popped and stored in the N and N+1 words of the entry. (The top level is put into the Nth word)

Text. Instructions

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
1	SETLP	[SN]	The top level of the stack is popped and stored in the LP.
1	RDLP		The LP is pushed onto the stack
1	DELL		The line pointed to by the LP is deleted
1	APPL		The line from the text buffer is appended to the entry at the end of the text group. LP is set to point to new line.

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
1	INSL		Same as APPL, except line is inserted before the line pointed to by LP. LP is set to point to inserted line.
1	CHGL		The line pointed to by the LP is deleted, and the line in the text buffer is stored in its place
1	LDL		The line pointed to by the LP is loaded into the text buffer. The LP remains the same.

V. Input/Output Commands

There are five devices in the present system which handle I/O:

- 1) Disk
- 2) 360/75
- 3) PDP 10
- 4) Teletype
- 5) Data Processing Unit (DPU) - PDP8/I Display controller

The instructions which control the flow of I/O between the program and these devices are described below.

Input:

A program expecting input from a particular device sets a transfer vector to specify where execution is to start when the input is received, and then issues a wait. This is done using the set of instructions described below. Note that if an address is not specified the input is flushed.

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
1	WAIT		Control is returned to Scheduler. The program will gain control again when it receives some input.
2	VECDPJ	(A)	The transfer vector for a joystick hit from the DPU is set to address A. If a joystick hit occurs, the x, y coordinates of the joystick hit are on the stack. (y coordinate is on the top level), and the program gains control at address A.
2	VECDPT	(A)	The transfer vector for a text line from the DPU is set to address A. If a text line is received, the text line is placed in the text buffer, and the program gains control at address A.
2	VECTTY	(A)	The transfer vector for teletype input is set to point to address A. If teletype input occurs the input line is placed in the text buffer, and the program gains control at address A.

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
2	VEC360	(A)	The transfer vector for input from the 360 is set to point to address A. When input is received it must be copied into an active file using a RCV360 command. (See below)
2	VECP10	(A)	The transfer vector for input from the PDP 10 is set to point to address A. When input is received, it must be copied into an active file using a RCVP10 command. (See below)
2	VECTMR	(A)	The transfer vector corresponding to "interval timer expired" is set to point to address A. The interval timer may be set using the SETTMR command. (See below). When the timer expires, the program gains control at address A. If A is not specified the timer is disabled.
1	SETTMR	<SN>	The top of the stack is popped and stored in the interval timer for this program.
1	RCV360		The input file from the 360 is stored in the file pointed to by the FP
1	RCVP10		The input file from the PDP10 is stored in the file pointed to by the FP.
1A	FLOAD	N1	The file whose name is in the text buffer between pointers P1 and P2 is loaded into the file pointed to by the FP. N1 is the extension number of the file.

Output:

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
2	SENDTTY	T	The text string, T, is printed on the operator's console.

The next group of output instructions deals with operations on files or parts of files. At present, there are three types of operations

- 1) Display - the file is displayed on the display screen at the user terminal
- 2) Local Storage - the file is stored locally on a mass storage device, currently a disk
- 3) Remote Storage - the file is stored in either the 360 library or the PDF10 library.

1) Display

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
1	DISPF		The file pointed to by the FP is displayed on the display screen
1	DISPB		The block pointed to by the BP is displayed on the display screen
1	DISPE		The entry pointed to by the EP is displayed on the display screen
1	DISPL		The line pointed to by the LP is displayed on the display screen
2	DISP	T	The text string, T, is displayed on the screen. The coordinates for the line are on the top two levels of the stack. The y coordinate is on the top level.

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
	DISPTB		The contents of the text buffer are displayed on the screen in the area designated as the keyboard text-line area.

2) Local Storage

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
1	FDEL		The file whose name is in the text buffer between P1 and P2 is deleted from the disk
1	FSAVE		The file whose name is in the text buffer between P1 and P2 is saved on the disk

3) Remote Storage

<u>Type</u>	<u>Mnemonic</u>	<u>Operand</u>	<u>Description</u>
1	SEND360		The file pointed to by the FP is sent to the 360 library
1	SENDP10		The file pointed to by the FP is sent to the PDP10 library

VI Pseudo Ops and Special Commands

Pseudo Ops

A = E A is assigned the value of E. E may be any expression involving + or -. If E is preceded by a /, A is assigned to be a name (label) for a stack level. Note that this pseudo-op may be used in conjunction with the POP instruction to pop the stack down to a previous level.

For example, consider a part of a program which marks the stack at one point, and then at a later time wishes to restore the stack to the previously marked level. The code would be as follows:

y = /0 Mark the stack

POP y Pop the stack to previous level.

TEXT T This instruction generates a word which contains the length in characters of the text string, T, followed by the text string stored in packed 6 bit ASCII. (Two characters per word.)

A DSV N The address A is defined as a stack variable and given the initial value N.

It should be pointed out that no variables are kept within the user's program. All variables are in the stack. This was done to make programs entirely re-entrant. (The MOVE instruction enables the user to reference any variable in the stack.)

Special Instructions

GRID $x(x_0, \dots, x_n), y(y_0, \dots, y_m)$

This instruction defines a grid which subdivides the display screen into logical segments. The $x_0, \dots, x_n$ in the GRID instruction represent vertical lines which divide the screen into $n-1$ sections. The $y_0, \dots, y_m$ represent horizontal lines which divide the screen into $m-1$ sections.

The parts of the screen in which these two groups intersect represent the inside of the grid. The segments inside the grid are numbered from left to right, top to bottom starting with 1. Anything outside the grid is assigned the number 0. See Figure 5 for a sample screen and GRID instruction. This instruction is used in conjunction with the SETGRD instruction. (The GRID instruction should have a label so that it may be referred to in the SETGRD instruction.)

SETGRD A

This instruction assumes that the x-y coordinates of a joystick hit are on the top of the stack. These coordinates are popped from the stack, and then used to determine which segment of the Grid, A, was referenced. The grid segment number is then pushed on the stack. If the hit was not inside the grid, a zero is pushed on the stack.

For example, if the joystick hit had the coordinates: $x = 20$, $y = 25$, and the SETGRD instruction referred to the Grid in Figure 5, then the coordinates would be popped from the stack, and three would be pushed onto the stack.

ADRTBL $A_1, \dots, A_n$

This instruction specifies a list of addresses to be used as an address table. It is used in conjunction with the BRTBL instruction.

BRTBL A

The top level of the stack is popped, and used as an index into the address table which begins at A. The program then branches to the location specified by the indexed address.

4. SWITCHING THEORY AND LOGICAL DESIGN

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

Works on transduction methods, MOS network synthesis, logical design by integer programming, and the problem to minimize the number of connections have been continued.

Optimal networks with ECL which minimize the number of NOR-OR gates, first and the number of connections, second, are designed, assuming that wired-OR may be used in networks. This work will be the master thesis of T. Kawasaki.

S. MUROGA

A Study of discrepancies in certain computational results (statistics for obtaining optimal NOR networks with 3 external variables using ILLIP with a feed-forward inequality formulation) contained in various reports and papers published by this group in the past was continued.

Investigations were also made into recently detected errors and irregularities in inequality formulations used in the past to obtain published data.

(J.N. Culliney)

During this quarter a thorough study was made of the generalized consensus and its application to boolean minimization. Several algorithms for this were developed.

(R. Cutler)

It was reported in the Quarterly Report for July, August and

September, 1973, that the experiments made on Gimpel's program were completed. Some time was spent this quarter documenting the computer outputs and programs. Also, the process of revising and correcting my masters thesis was begun. Finally, the reading of the papers describing the network transformation programs (see previous Quarterly Reports) was continued. However, the emphasis was changed from considering how to combine the various procedures into an effective network design system to implementing arbitrary fanin and/or fanout constraints into some of the transformation programs.

(K. Hohulin)

Since October, I have run a program which minimizes the number of gates and/or connections for a given network. Using GCMC (Gate-Connection Minimization Criterion), 619 4-variable functions were tested, and, among these, 229 networks were rerun using CGMC (Connection-Gate Minimization Criterion) to obtain the networks with minimum numbers of connections.

Recently, our group has begun to study the possibility of obtaining "near optimal" networks for a given function by transduction procedures under fan-in and fan-out restrictions. My task is to modify the Error-Compensation procedures, i.e; incorporate the consideration of the fan-in and fan-out restrictions into the original Error-Compensation procedure. Now I am studying papers on the properties of permissible functions and network specification, and papers on network transduction by the Error-Compensations method.

(J.K.C. Hu)

During the period, several zero-one integer linear programming formulations for solving various types of minimization problems in the

logice design of negative gate networks have been worked out. These formulations have their practical application in the design of MOS integrated circuits since an MOS cell can realize any negative function, and therefore MOS circuits can be treated as negative gate networks. Using the computer programs developed for solving general zero-one integer linear programming problems, these formulations can be easily solved.

(H.C. Lai)

Publications

C.R. Baugh, "Chow Parameters in Pseudo Theshold Logic", 14th Annual Symposium on Switching and Automata Theory, Oct. 1973, Iowa City, Iowa.

J. Culliney. "Use and Descriptions of the CALCOMP Program to Draw Networks of Logic Gates". UIUCDCS-R-73-580, June, 1973, 44 pages.

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)

5.1 PROGRAM ANALYSIS

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

An experiment using a simple folding scheme in the evaluation of general linear recurrence systems and uniform linear recurrence systems has been performed in order to investigate the relations among the number of processors, speedup over a serial machine and the parallel machine efficiency. The result is still being analyzed. However, the data has shown that the parallel algorithm which we developed for solving these problems demonstrates that we can design or use a parallel machine with reasonable efficiency and higher speed than any known serial/parallel method. The details of the result will be summarized in a future report, and the research will continue to probe into broader application problems as well as theoretically unsolved problems.

FORTRAN Program Analysis - (R. Towle)

Summaries of analyzed programs were completed. IF statements in DO loops were classified as to function and also to form. Techniques to eliminate them were also investigated.

Numerical Stability of Parallel Computation - (K. Morgan)

During the last quarter the numerical stability of alternative orders

COBOL Analysis - (L. Chan and R. Strebendt)

Work is continuing on the problem of speeding up file processing programs through the concurrent processing of input records. A sample of about forty programs written in COBOL have been prepared for analysis. A scanner to aid in the analyses is now being debugged.

As a result of early manual analyses, a number of programming conditions were identified which severely hinder speedup attempts. Methods of handling several of these problems have been found. Algorithms for virtually all of the compiler steps that would be needed to implement our techniques have been developed.

5.2 FILE PROCESSING

Information Retrieval Computers - (W. Stellhorn)

Work is continuing on the analysis of a parallel processing computer for information retrieval from inverted files. Preliminary simulation studies indicate that the proposed machine can conduct a large search, involving 70 terms, in about 10% of the time required by a conventional computer. They also indicate that a large number of short searches (three terms each) can be conducted in parallel without seriously degrading performance.

Current efforts are aimed toward completing the details of a design study intended to show the feasibility of the proposed machine and toward refining test results, especially in the area of multiple simultaneous searches. Algorithm optimization studies are also in progress.

File Processing Software - (B. Hurley)

During the first two months of this quarter, work centered on the preparation of the new data bases for future use by the Information Retrieval Group. A new data base containing business abstracts is completely edited and ready for use. A preprocessor was coded for the Wisconsin State Statutes data

base to alleviate problems which our current editor could not handle. Work on the final version of these documents is now in progress. In January, emphasis shifted to becoming familiar with and improving algorithms used by the Information Retrieval Computer Simulator.

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

The two DIVA disk drives arrived this quarter and the PDP-11 Disk Operating System (DOS V4) was brought up on one 11/40. Modifications to the disk driver were made to control two drives and remain compatible with the other DIVA user in the department (thus permitting program exchange and occasional use of their printer and DEC-tapes by swapping disk packs). A 32-bit system clock similar to the DEC KW-11 was defined and is ready for logical design. This clock will be used to measure system activity, keep time-of-day, and time out the 11/40 end of the 360/75--11/40 data line. Software for both the 360/75 and 11/40 ends of the parallel data adapter (PDA) has been designed and coding is in progress. Debugging will be delayed until the PDA and possibly the clock are completed.

During this quarter, the user language parser was completed and debugged. During the next quarter, several changes to the language syntax, and thus the parser, are planned. A file handler will also be written during the next quarter.

Also, the major routines for the PDP-11 information retrieval system were written and debugged. Currently a user can search 64 documents. The only routine which has not been written is one to save a user's comments and the text of his queries. This should be completed by mid-February.

During the next quarter, some additions will be made to the retrieval language to improve output capabilities and to facilitate browsing. Also, design of experiments with users will begin.

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

Most work has been diverted toward the installation of the new PDP-11/40 system. The memory on one of the processors was increased to 24K, with the other processor having no memory, to allow DEC's DOS to run.

The dual disk system from Diva arrived and was connected with no difficulties. A printer from Centronics, core from Datacraft, and terminal interfaces from DEC were ordered for installation next quarter.

The designs were completed on an interface to the IBM System/360 through the 2701 PD4, an interim memory address translation unit, an interface for the new printer and the card reader, and a new wire-wrap utility card.

Publications

D. Kuck, P. Budnik, S-C. Chen, E. Davis, Jr., J. Han, P. Kraska,
D. Lawrie, Y. Muraoka, R. Strebendt and R. Towle,

"Measurements of Parallelism in Ordinary FORTRAN Programs,"
Sagamore Computer Conference on Parallel Processing,
Syracuse University, pp. 23-29, Aug. 1973.

D. Kuck, "Multioperation Machine Processing of Ordinary Programs,"
invited paper, to appear in First International Symposium
on Computers and Chinese I/O Systems, Taipei, Taiwan, Aug.
1973.

D. Lawrie, "Memory Systems for Parallel Array Processors," Proceedings
of the Eleventh Annual Allerton Conference on Circuit and
System Theory, Allerton Hse., Monticello, Ill., pp. 568-576,
Oct. 1973.

6. COMPUTER SYSTEMS ANALYSIS

The goal of this research is the development of analytical tools for system modeling and analysis of real time computer networks. Research this quarter has been aimed at surveying system performance evaluation in existing and proposed computer nets.

Network Performance Evaluation (S. A. Mamrak)

Although the computer network concept developed in an unrevolutionary manner, proceeding logically and in an orderly way from the development of highly sophisticated single processor systems, the performance evaluation techniques employed in single processor and geographically distributed multiple processor computer systems differ radically. Performance evaluation of single processor computing systems is characterized by a hodge-podge of performance goals and performance measurements. The most significant convergence of thought among single processor systems' analysts is agreement that what is required is a quantitative methodology on which to base analysis of real system data for model formulation and validation. Performance evaluation in networks, on the other hand, where it has been present, has been characterized by a careful development of analytic and simulation network models, generally supported by data analyzed using optimization and statistical techniques. Thus far the full advantage of these system evaluation methodologies has been enjoyed almost solely by the ARPA network research group. The modeling and measurement tools are general enough, however, to be widely utilized in the development and modification of any computer network system.

An examination of existing models and measurements in computer network systems reveals several trends. Analysis based on queueing theory has been anchored in node-by-node analysis, assuming independence of the various network nodes. This approach works very satisfactorily for a limited set of network phenomena. Simulation has been successfully used, but can become prohibitively expensive when detailed representations of the network system are required. Optimization techniques have been effectively transferred from network flow theory and are working well to yield specific design parameters. Actual system measurements, analyzed using statistical techniques and used to improve queueing and simulation models, have been relatively neglected. (This neglect may be due in part to the unavailability of tools for making desired observations of dynamic systems and of statistically significant test environments.) Finally, although sophisticated performance evaluation tools are generally available, they have been applied almost solely to the ARPA network.

The results of this survey are fully documented in a forthcoming Department of Computer Science report entitled "Performance Evaluation in Computer Networks: A Survey."

(E. K. Bowdon)

7. ASPECTS OF COMBINATORIAL COMPUTING

Recent work includes study of networks of queues, stacks, and dequeues, sorting and selection networks, and scheduling algorithms.

1. Networks of queues, stacks, and dequeues (J. Koch)

Work was completed this quarter on computing permutations with networks of queues, stacks, and dequeues. An efficient method for determining the characteristic set of a 1-level network was discovered. (The characteristic set of a network is the smallest set of permutations that the network cannot compute and that must appear as subcomputations of all longer permutations that the network cannot compute.)

2. Sorting and selection networks (J. Bitner)

Research was carried out on the analysis of non-adaptive sorting networks. A new (non-recursive) description of Batcher's odd-even-merge networks was discovered. I am presently working on the development of a class of networks that was conjectured to be superior to the Batcher networks.

3. Scheduling algorithms (N. F. Chen)

A simple algorithm for scheduling jobs with identical execution times was studied. We were able to show that:

1. For an n -processor system, if the set of immediate successors of any n jobs contains no more than n independent jobs, then this algorithm yields an optimal schedule.

2. For a 2-processor system

$$\frac{\omega_l}{\omega_0} \leq \frac{4}{3}$$

and for a 3-processor system

$$\frac{\omega_l}{\omega_0} \leq \frac{3}{2}$$

where ω_0 is the completion time when an optimal schedule is used and ω_l is the completion time when the schedule produced by this algorithm is used.

4. Scheduling algorithms (S. Dhall)

Algorithms for obtaining optimal or close to optimal schedules for the execution of jobs not necessarily of identical executing times were studied. Results have been obtained for a special case in which the execution time of a job is either 1 or k (integer).

Publication

Liu, C. L., "A Combinatorial Study of Some Scheduling Algorithms" Proceedings of the 2nd Annual Texas Computer Conference, November, 1973, pp. 40-1 - 3.

8. THEORY OF DIGITAL COMPUTER ARITHMETIC

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

8.1 Function Evaluation Techniques

During this quarter several problems related to the parallel methods of function evaluation, based on linear combinations of selected precomputed constants, were investigated. Different approaches in defining a basic set of constants were tried in order to satisfy given precision requirements and still retain a non-iterative selection procedure. An already investigated method, which is based on a minimax solution, can be applied only on a reduced argument range. Among other possibilities, decomposition of a given function into odd and even auxiliary functions defined on half of the argument range proved inconvenient. The other strategies, under current investigation, include modifications of basic constants using low-precision operations and their selection on the basis of an argument, modified using some simple transformations.

(M. D. Ercegovac)

8.2 Implementation of a High Speed Calculator

This quarter work continued on the DeLugish calculator. As described in the last Quarterly Progress Report, the calculator is based on the method of continued products as developed by DeLugish in his thesis. It is designed to do multiplication, division, exponentiation, and find logarithms on the same hardware. The logic design has been

developed and minimized. At this point in time one register bank has been built and tested. The parts have arrived for the barrel shifter and work is continuing on the remaining hardware. Things look promising at this point for getting all of the hardware required into two EXCEL boxes.

Future prospects are for developing this elementary fixed point calculator into a floating point one which will find square roots and trig functions in addition to the previously defined functions. Ultimately, it could be developed into a hardware arithmetic unit which could be attached to the PDP-11 via a busing system.

(M. J. Irwin)

9. The Computer Science-PLATO Project:
review and guidelines.

1. The project and its goals

Over the past year and a half, the Department of Computer Science has been heavily engaged in a project to develop on PLATO IV an instructional system capable of assuming a large part of the teaching load in all CS-100 level courses.

The major goals of the project are:

- the system should be useful as a supplement to classroom instruction in any introductory computer science course -- no matter what programming language is taught, what examples of computer applications are discussed, and what the instructor's philosophy is about how programming should be taught
- the system should be able to provide at least half of the instruction in any one of our own introductory CS courses (currently CS 101, 103, 105, 106, 107, 121)
- the system should be usable without any instructor at all by anybody having access to a PLATO terminal who wants to learn on his own the beginnings of computer science.

I like to summarize these points by saying that our instructional system should be able to operate in a partially automated mode (with the degree of automation determined by the instructor) as well as in a fully automated mode.

These goals have caused us to organize the instructional system as a collection of relatively independent and self-contained components. We do not want to take one of our existing CS courses and put it on PLATO as it is. We do want to turn a PLATO terminal into a rich environment where you have at your fingertips many useful things for learning about

computer science. It is the student and his instructor who decide which ones of these things they want to use.

Last but not least, the project has one more important goal:

- to serve as a stimulating environment for computer science research in a variety of areas: compilers, information systems, artificial intelligence.

This last point may deserve some explanation, since the misconception is widespread that lesson writing is a routine activity. It can be, if one creates poor lessons. It can also be a task as challenging as you wish to make it, if you view a lesson as an interactive program that has a certain domain of knowledge, and is able to communicate with students about the knowledge it has.

2. The main components and their current status

A. The library of lessons

We have about 50 lessons that are in some sense "operational", but most of them need a lot of improvement. We classify them as follows (approximate number of lessons in parenthesis)

- a) Introductory lessons
- b) Programming languages: PL/1 (15), FORTRAN (10), BASIC (2), Snobol (2).
- c) "Minilanguages" (artificial little languages designed to illustrate one particular programming concept) and language-independent programming concepts (10)
- d) Applications: numerical analysis (8), others (6)

B. Language processors (interactive compilers/interpreters)

- a) PL/1: preliminary version is working

- b) FORTRAN and BASIC: program-entry and -editing, and syntax analysis part is working. Run-time system under construction

C. Conversational information and advising system

Purpose: to advise students in their choice of lessons, based on their goals and past performance. This is our approximation to "Computer-Managed Instruction". It consists of three main parts:

- a) a catalog of all lessons with a rich description of relationships among them: in design stage
- b) a translator from English into an internal representation of requests: completed
- c) a request processor to interpret this internal representation: in design stage

D. Communication system

Purpose: to allow communication via PLATO among students, instructors, authors and the management of the instructional system. Preliminary version is working.

3. Use of the instructional system to date

The system has been used in several courses. The most extensive use to date involved half the class (60 students) in CS 121 in the fall 73, who learned PL/1 on PLATO. This event was used to try out our procedures for a controlled experiment to evaluate the effectiveness of PLATO instruction. In the spring of 74 we have started to use the instructional system in an auxiliary mode in several CS 100-level courses, and also in CS 211.

4. Priorities for 1974

First, we must concentrate on a few parts of our system and make them as good as possible. Compilers, the information and advising system, and a few lessons that can be justified as artificial intelligence research, are the most likely candidates. There is a need to demonstrate what CAI can do at its best.

Second, we must make every effort to use the system as much as possible. We need feedback from students and instructors, and CS instructors should become acquainted with our systems.

Third, we must improve the lessons we have. Development of new lessons should have a low priority at this point.

Fourth, we should attempt to cooperate with computer science departments at other schools that have PLATO terminals. We should aim at a system that is used for computer science instruction over the entire PLATO user community, not just at the U. of I. Also, the library of lessons we would like to have is so large that any help we can obtain is most valuable.

5. A major problem

The size and complexity of the instructional system we are aiming at is beyond what can be reasonably managed and maintained by a staff of faculty members and assistants, each of whom can devote only part of his time to this project. A nucleus of full-time professionals is necessary to provide continuity and consistency.

Let me compare the work required to set up our instructional system to that involved in more conventional activities.

Writing a library of 100 lessons can be compared to writing 100 tutorial papers up to publication standards. But there is more to programming a lesson than there is to writing a paper on the same subject. Not only because of the well-known phenomenon that it is much harder to explain something to a computer than to people, but primarily because, if one wants to create a good lesson, one always does many things that one could not do on paper (such as incorporating a model that the student can manipulate). These things almost always require fairly intricate programming.

If the 100 lessons have to fit into a coherent scheme, the project assumes the characteristics of writing an encyclopedia. But again, on a CAI system one wants to implement much more elaborate relationships among the lessons than one would find among the different sections of an encyclopedia (E.g.: does the student have trouble with writing conditions in IF-statements? Check to see if he ever took a lesson on boolean expressions. How well did he do there? Branch him off to an appropriate section of that lesson). The task of writing n compatible lessons grows much faster than linearly in n . Even the day-to-day maintenance of lessons that are already in good shape is a nontrivial task. When a new lesson gets written, should the old ones refer to it? On top of these inherent difficulties, there are such bothersome facts as: PLATO keeps changing fairly rapidly; frequently authors of existing lessons are no longer around when one would like to make a change in their lesson, while other authors want to keep changing their lesson when we would like it to remain constant.

In the long run we have to aim at a system that works like the publishing industry. At a certain moment a formal agreement is made to the effect that the rights to the content of the lesson pass from the author to the management of our instructional system. This will impose additional

work and responsibility on the management, but I see no other way to guarantee that a system as complex as ours will run smoothly.

J. Nievergelt

10. MACHINE PERCEPTION AND INTELLIGENCE
(AQVAL Project)

1. Theory and Implementation of Variable-Valued Logic Systems
(Michalski, Yuan, Yip, Peterson, Hamilton)

Two new algorithms have been developed by Michalski:

1. An algorithm for direct minimization of an arbitrary VL_1 formula according to a lexicographic functional (specified by a user according to his needs).
2. An algorithm for the synthesis of uniclass covers based on generation of stars which satisfy an assumed density threshold.

Both algorithms are being implemented in PL/1, the first algorithm by Yip and the second by Yuan.

Various properties important for applications of the current version of AQVAL/1 program (Version 7) have been investigated, specifically the dependence of the execution time and memory requirements on the number of variables, sizes of their domains, and sizes of learning event sets, and also the investigation of the problem-independent and problem-dependent core sizes.

A program CHECK has been developed for estimating the amount of core required for a given problem.

Work has been started on the development of a new, more convenient, user-oriented format of the input and output data for the current (V. 7) and future (V. 8) versions of AQVAL/1.

2. Applications of VL Systems

(Michalski, Yip)

AQVAL/1 was used for developing formulas for detection of the cancer of colon based on data obtained from Mercy Hospital.

A paper on the AQVAL program and its applications to a medical diagnosis problem (detection of the cancer of the liver or of the pancreas) and to a non-uniform texture discrimination problem has been presented by Michalski at the First International Conference on Pattern Recognition, Oct. 30 - Nov. 1, Washington.

Work was carried on the application of AQVAL/1 for developing formulas expressing a relationship between the soundness of rocks and their various easy to measure parameters.

Publications:

Michalski, R. S., "AQVAL/1--Computer Implementation of a Variable-Valued Logic System and the Application to Pattern Recognition," Proceedings of the First International Joint Conference on Pattern Recognition, pp. 3-17, Washington, D.C., October 30-November 1, 1973.

11. STATISTICS

Regression equations have been developed for estimating the size of an eigenvalue λ from a correlation matrix with squared, multiple correlations as the diagonal elements. There is a separate equation for each ordinal eigenvalue λ , depending on N (the sample size), n (the number of variables), and m , the ordinal number of the eigenvalue. Each equation has the form: $\log \lambda = a + b_N \log(N-1) + b_n \log(\frac{n(n-1)}{2} - (m-1)n)$. The results of the sampling studies are presented in the table below. Work is continuing on extending the table and also on an attempt to discover some model which will put all eigenvalues in one equation.

Root	a	b_N	b_n	R(multiple correlation)
1	.426	-.625	.381	.9994
2	.223	-.637	.454	.9982
3	.051	-.651	.523	.9976
4	.043	-.649	.509	.9985
5	-.031	-.654	.536	.9980
6	-.002	-.670	.528	.9964
7	-.085	-.657	.540	.9979
8	-.075	-.678	.548	.9953
9	.086	-.712	.506	.9870
10	-.081	-.666	.520	.9959
11	.047	-.706	.500	.9941
12	.302	-.765	.443	.9839
13	-.263	-.687	.602	.9986
14	-.071	-.741	.573	.9950
15	.276	-.806	.484	.9910

(R. G. Montanelli, Jr.)

12. MESH Project First Annual Report

January 1, 1974

First Annual Report

Project Mesh

NSF Contract GJ 36265

As our quarterly report for the last quarter of 1973 we are submitting the annual report less the appendices. Copies of the appendices are available for anyone interested in further details.

Professor D. B. Gillies
(Principal Investigator)

Professor J. W. Liu

G. Chesson
S. Davidson
J. Hart
M. Ketelsen
P. Krabbe
J. Krishnaswamy
J. Laprade
E. McClary
T. Miller
C. Nordheden
B. Raphael
J. Renshaw
I. Stocks
W. Tao

12.1 Introduction

This report summarizes the progress made in the year 1972-1973 in the MESH project. In this project, we are concerned with the design and application of centralized minicomputer networks which contain a diversity of computing elements. To date, small mass produced minicomputers offer features such as alterable microprogram, elegant interrupt, stack and list processing capability at low cost and are as fast as larger machines. For this reason, we believe that a network of minicomputers each micro-programmed to be a special-purpose node can be designed to perform non-numeric processing such as data base management, language translation, list processing, etc. more cost effectively than a large general-purpose computer.

A great deal of developmental effort has been directed towards the design of minicomputer networks.[†] Minicomputer networks with vastly different architectures have been designed and implemented with specific applications and design objectives in mind. Despite their differences, these minicomputer networks clearly demonstrate advantages, such as natural system reconfiguration, orderly growth capability, functional isolation and modularity in software and hardware, graceful degradation, high system reliability and availability, over large general purpose computing systems.

Although there have been a large number of minicomputer network installations, the problems pertinent to minicomputer network design such as processor-to-processor and processor-to-memory linkage, distribution of fast memory among processors, the structure of basic building blocks in the network, the tailoring of network architecture to network usage, hardware/software/firmware tradeoffs, effective algorithms for resource

[†]A survey of minicomputer networks is included in [1].

management in network environment, etc. have not been studied systematically and thoroughly. Furthermore, studies in these areas have been mostly for networks composed of general-purpose building blocks. For example, networks which consist of several identical processors connected to primary memory modules and devices through a matrix switch have been studied [2-4]. Such a network can function as several independent minicomputers by allowing the interconnections between processors, primary memory modules, disk drives and other devices be made for relatively long time periods. Each active process in the system has its own processors and secondary memory [3]. On the other hand, if the connections between processors and memory modules are maintained only for the transmission of one data word, the system functions as a multiprocessor system [2]. The results obtained in these studies on design limitations of the matrix switch, desirable characteristics of the building blocks, memory organization and contention, system evaluation, and error recovery, etc. are also summarized in [1]. Investigations of global structure of the network [5-8] and methods for control of large networks [9,10] have also been confined to networks composed of general-purpose building blocks.

As proposed in [11], our attention has been directed towards the design of a general-purpose computing system which will have as a nucleus for data processing, storage and retrieval; a centralized network of processors and memories, most of them small and cheap. Each processor in the network will be designed to perform special tasks and thus can be considered as a special-purpose node in the network (or in a subnetwork). For example, one might be designed for compilation, one for sorting and merging, and one for numerical calculation. We have been concerned with finding cost

effective designs for special-purpose nodes and subnetworks in the system as well as studying various trade-offs in the design of the global network structure. Results obtained to date will be summarized in Sec. 3. In Sec. 2, a development facility designed to be a general test bed is described.

12.2 Hardware and Software Support

A development facility was designed and implemented in the past year. The primary function of the development facility is to serve as a general test bed so that evaluation of different network designs can be performed experimentally. For example, the structures of nodes and subnetworks in the centralized minicomputer network have been investigated. By having an appropriately designed test bed, various designs of the nodes and subnetworks can be implemented, debugged and tested in a simulated network environment. Since the nodes in the network may be microprogrammable minicomputers, the development facility includes a microlevel assembler and simulator. Furthermore, facilities such as writable control storage, various types of communication links, etc. are included to allow implementation of real-time test environments. Since research on network control procedures, traffic rates, data/control path organization, etc. may require modeling tools for simulation and evaluation, the development facility also includes the necessary computing equipment and data base to facilitate network modeling.

The development facility is interactive so that it can provide the user a complete facility for the control of experiments and preparation of program, data and reports. It can support several users simultaneously. The particular physical configuration of the development facility was chosen because it can be implemented with the least cost both in time and effort. In this section, we will describe the physical configuration of the development facility. The software and firmware developed to date will also be discussed.

12.2.1 Physical Configuration of the Developmental Facilities

Physically, the developmental facility is a closely coupled network containing three independent nodes shown in Fig. 1. Each of the two PDP-11 processors is connected to its own primary memory, disc storage units and I/O devices by a Unibus. Among the four SUE processors connected to the Infibus, one processor has been modified. Under program control, it can operate either with the standard SUE microprogram counter and ROM microprogram and literal store or with a modified microprogram counter and a writable control store. The modified microprogram counter with its 32 word hardware stack and the writable control store shown in Fig. 2 are described in detail in [12]. These modifications of the SUE processors will allow us to reconfigure the SUE system to emulate other minicomputers and thus provide us with the versatility necessary in experimental evaluation of different node architectures and network configurations.

The network implemented so far is linearly connected. The two PDP-11 systems are connected via a full duplex channel which allows data transfer at an average rate of 2M BAUD. This duplex channel runs under program control without endangering the operation of either bus in the event of failure of the other.

The Unibus 2 and the SUE Infibus are connected via three separate links[†] as shown in Fig. 1. The memory controller, which contains the hardware for memory segmentation, will allow access from both buses to a 2-port 64K word common memory. Memory segmentation is controlled by the content of registers accessible only to the control bus. The memory

[†]Since these communication links are between the same two terminals, they share some of the hardware such as bus drivers and transmitters.

Fig. 1 Developmental Facility

- R0 - status
- R1 - control and store address
- R2 - ST manipulation
- R3 - unassigned
- R4 -
- R5 - { ports to RAM (1K microwords 38 bits
microprogram expandable to 4K
store microwords 48 bits)
- R6 -
- R7 - port to RAM (256 8 bit words
literal store expandable to 1K
16 bit words)
- S - microprogram counter (1 word 12 bits)
- ST - push down store (32 words 12 bits)
controlled by two
additional bits in
the microword called
the "P" field

Fig. 2 Details of the Modified SUE with WCS

segmentation scheme used allows mapping of variable sized segments of 8K words or less from name space to physical memory space. Up to 64 segments can be handled by this memory controller.

The interrupt control system is another control and communication link between the PDP-11 system on Unibus 2 and the SUE system. It contains a set of registers whose contents are software modifiable. These registers contain a set of special address ranges and a corresponding set of interrupt vectors. The interrupt control system is designed to recognize addresses in specified ranges. When an access to an address in one of the specified ranges is made through the Infibus, no memory cycle will be initiated. Instead, it causes an interrupt on Unibus 2 at the interrupt vector given by the contents of the corresponding interrupt vector registers. That is, with the help of the PDP-11, Infibus I/O requests may be either passed to real devices on the Unibus, or to the file system.

The trans-bus access system permits the PDP-11 processor on Unibus 2 to read from or write into any address on the Infibus. Through this system, the PDP-11 processor can interrogate, modify and initialize various status words and thus starts or terminates a process without the aid of a processor on the Infibus. A technical report is being written to describe the details of the physical implementation of the memory controller, the interrupt control system, the trans-bus access system and the full duplex channel between the two Unibus.

12.2.2 Software Development

The structure of the network operating system designed and partially implemented is shown in Fig. 3(a). The system provides general stand-alone

facilities as well as communication and control mechanisms necessary for access to the network environment. The various subsystems in this network operating system are designed to be implemented separately in order to provide stand-alone usage of the subsystems during the development period.

The basic modules of this operating system are the node-based operating systems (NBOS) which reside in the three stand-alone physical nodes of the network. To date, only the NBOS1 and NBOS2 supported by the two PDP-11 systems have been designed and implemented. NBOS3, which will reside in the SUE multiprocessor system is currently being designed.

The internal structure of NBOS1 and NBOS2 is shown in Fig. 3(b). Each node-based operating system furnishes standard high-level device I/O, the file system and general purpose utility routines in the standard disk operating system (DOS) supplied by the Digital Equipment Corporation. In addition, it contains a multi-task interface (MTI) which handles time-slicing and content switching, manages system resources, and furnishes synchronization primitives to control cooperation between independent processes. Thus, it is possible to simulate several virtual nodes of a network within a physical node. Currently NBOS2 is a multi-programmed interactive system which provides a text editor, a file utility package and a basic monitor for program loading. NBOS1 is a batch processing variant of NBOS2. It is set up for background spooling of batch-processing I/O devices.

The communication and control system (CCS) shown in Fig. 3(c) can be considered as a software matrix switch linkage which will establish a communication link between any pair of virtual nodes in the network. It consists of three identical submodules, CCS1, CCS2, and CCS3. Each submodule resides in the corresponding physical node. In Fig. 3(c), the connections

Fig. 3(a) Distributed Operating System

Fig. 3(b) NBOS Structure

Fig. 3(c) Distribution of CCS Modules

Fig. 3(d) Structure of Local CCS

between the node-base operating systems and their corresponding communication and control submodules represent software linkages. The lines connecting the communication and control submodules represent one or more physical communication channels.

The internal structure of the submodules is shown in Fig. 3(d). The command processor maintains control and communication information on local processes as well as the local position of the virtual network connection matrix. Data needed for control and communication are maintained only for those processes which have joined the network through the local interface. When the two communicating virtual nodes are in two different physical nodes, the communication path across the hardware link between the two physical nodes is established through the channel drivers. The details of the CCS system will be documented in another report.

PASCAL has been chosen as a high-level language for writing system programs. It has been demonstrated that programs written in PASCAL are transportable. It is a simple, elegant, powerful and efficient language and sufficiently concise to run in a small amount of core. The insistence on precise definition of every quantity used and of every operation performed allows many compile-time checks for semantic meaningfulness to be made. Since the language has very powerful data structuring abilities, the necessity of defining precisely the set of values each variable may take is not a hardship. In a multi-processor environment, even the most trivial errors can have devastating repercussions and the ability of avoiding a large class of errors is worthwhile. Furthermore, a compiler for a large subset of the language can be brought up in a short time.

A bootstrap version of PASCAL compiler has been designed and implemented. It consists of two passes. The first pass, which is the

syntax analyzer, performs a complete syntax analysis on the source program, attempts to repair obvious errors, creates and formats a listing, and creates a file of intermediate code. This intermediate code in a pseudo-assembly language format represents an irredundant form of the source program. In the second pass object code is generated from the intermediate code. Both passes are coded in Macro-11 with a suitable file of Macros defining the intermediate language. In order to speed up the second pass, it is being re-coded in PASCAL itself. The compiler is described in detail in Appendix A.

After code is generated, it is linked with a run-time package, which currently contains I/O and file-oriented operations, storage allocation and deallocation, a preliminary form of an external procedure loader, and simple job initialization routines. At present, it is possible for one job to initiate another, but not to exercise control over it. A simple initiator has been written in the intermediate language, using features in the run-time routines but not yet in the language. Up to eight jobs have been run at one time. Switching between jobs occurs either on expiration of a time-slice, or on initiation of I/O. Since the run-time package contains a loader, it can be made to overlay the less used sections of itself (such as software multiply-divide routines for the PDP-11/20).

12.2.3 Firmware Debugging Facilities

To date, all microprograms for the Lockheed SUE have been written and debugged using a simulator for the SUE. This interactive simulation program has been written in PDP-11 Assembler Language to run on the PDP-11. It supports disk files, card reader and line printer I/O, while the user communicates with the program through a console. Microprogram debugging

facilities such as breakpoint, dump, and trace are available. The simulator is compatible with the output of the SUE Microcode Assembler by simply linking the two together with the Linker under DOS. The complete description of the simulator is given in the user guide to the simulator (Appendix B).

A Lockheed SUE microassembler called MICROASM which runs on the PDP-11 has been developed. It can be used alone for assembly, or used to generate the input for the Lockheed SUE microinstruction simulator discussed above. The SUE microcode may be specified by mnemonics or by assigning values to specific fields of the microword. The MICROASM has been and is still being used successfully as a debugging aid for the Lockheed SUE microprograms. To provide error detection capability and improve assembly time required, an improved assembler called SMALL (SUE microassembler language) has been written in PASCAL on the PDP-11. A SMALL manual is included in Appendix C.

12.2.4 Simulation Facilities

As discussed in Sec. 2.2, the communication and control system together with the node based operating systems will allow us to simulate networks and multiprocessor systems of various architectures. In particular, it is possible to simulate pipeline and array systems, hierarchical networks as well as other network configurations.

The SUE system with the writable control store will not only provide us the additional flexibility in simulation of various node structures but also allows the implementation of a real time test environment. In anticipation of their need, emulators of a number of minicomputers will be implemented. A PDP-11/45 emulator has been microprogrammed for the Lockheed SUE minicomputer

and has been debugged using the SUE Simulator on the PDP-11 [13]. The emulator includes the entire PDP-11 instruction set except for the optional floating point instructions. An emulator for a 32 bit computer, the LGP-21, has been written in SUE microcode to evaluate the feasibility of using SUE to emulate machines with word length longer than that of the SUE [14]. The LGP-21 was chosen because it is one of the simplest 32 bit computers ever built. Thus the task of emulating it is a relatively short term project, and the difficulties in implementing a longer instruction word can be more easily examined.

The basic structure of many minicomputers are the same with their differences only in parameters such as number of registers, methods of interrupt handling, word length, etc. An emulation program which accepts these parameters as input and emulates a virtual machine of specified characteristics will be a valuable research tool in the study of networks. Work has begun in finding a minimal set parameters that will serve to specify the differences between various minicomputers.

12.3 Current Research

Several problems pertinent to the design of minicomputer networks consisting of a large number of special purpose nodes have been studied during the past year. In particular, we have studied the structures of special purpose nodes and subnetworks designed for high speed compilation, data compression and decompression and other non-numeric processing functions such as merging/sorting, etc. The designs of several types of special purpose nodes have been completed or initiated. A study of the trade-offs between possible implementations of these building blocks is being made.

Problems on the global structure of the network investigated include the evaluation of various process communication and control mechanisms in a minicomputer network environment in terms of flexibility and system overhead requirements [1]. Algorithms for resource management and job scheduling have been analyzed and evaluated for their suitability in a large heterogeneous system. The PASCAL language with extensions has been adopted for writing system programs. A compiler for PASCAL has been designed and implemented and is currently available to users of the development facility.

In this section, we shall discuss the results obtained to date for each of these problems.

12.3.1 Special Purpose Processors

In a network consisting of special-purpose nodes, any particular node may be a stand-alone minicomputer system.

To gain experience in firmware design several special-purpose nodes have been designed and implemented in firmware. A data compression/decompression node using the Lockheed SUE microprogrammable processor with all algorithms coded in firmware has been designed and debugged [15]. This node, shown schematically in Fig. 4, will be a subsystem in the large data base provided in the development facility. Placed in the I/O path, it functions under program control to either compress or decompress the data or leave it unaltered. In order for it to be insensitive to the data structure, the Huffman coding algorithm with user-specified maximum character string length was chosen. To determine the Huffman code for a given text, one first collects the statistics of occurrence of the chosen patterns in the text and establishes the frequency table and code table which define the coding

Fig. 4 Compression/Decompression Node

algorithm. Hence, the coding algorithm for data compression is table driven, and changes in statistics of occurrence of the patterns can be systematically handled by changing the code table. A software package was used to collect statistics of various patterns in the data needed for the construction and updating the code table. This package is run when the file is first compressed. It can be rerun if the text content changes to maintain code efficiency. Several simulation runs have been made. It was observed that the typical compression ratio for source code in PDP-11 MACRO assembler language was about 40%.

As discussed in Sec. 2.4, two emulators, PDP-11/45 and IGP21, have been implemented. In addition, the feasibility of a general emulation program which will accept parameters specifying a given minicomputer architecture and will return a virtual machine with the specified architecture is being studied.

12.3.2 Subnetwork Design

In the performance of complex tasks, such as language translation, more than one processor may be used efficiently to increase throughput and to achieve higher reliability and availability of the node. Hence, a node designed to perform a complex task may be a subnetwork of processors and memories. In these subnetworks, high system efficiency can result by matching the system structure to the characteristics of problems encountered. For example, array systems such as ILLIAC IV are suited for applications which require manipulation of multidimensional data, e.g., matrix multiplication. In these data manipulations, operations on the data elements are similar and can be performed concurrently. On the other hand, in applications where most operations can be performed in a sequence of steps, pipeline systems are suited.

We are concerned with finding the dependence of subnetwork design on the type of problem. In particular, a problem may be characterized by the data structures used, the type of operations performed on the data and the frequency of changes in the data structure or contents. We have chosen language translation, list and table processing, data base and array operations as characteristic problems.

The design of a high speed compiler subnetwork is being carried out and evaluated in depth. The process of compilation can be split into a number of sequential passes over the input data stream. The output data generated in each pass are the input data for the next pass [16]. If a processor is dedicated to each pass then the compilation processors can be viewed as a pipeline system. Communication between different processors is achieved through a combination of shared memory and high speed data links.

Our attention has been confined to the design of a pipeline compiler for the language PASCAL. It is desirable that the division of labor among the various passes be such that the time spent by the different processors in the pipeline are as equal as possible. Based on the run time measurements done on the PASCAL compiler described in Sec. 2.2, a preliminary division of labor has been drawn. The tasks performed by processors in the different stages are shown in Fig. 5. The first five stages of this compiler perform those tasks done in the first pass of the bootstrap PASCAL compiler in Appendix B. Each stage will be implemented using a microprogrammed processor. The distribution of local and shared memory, the choice and implementation of the primitives in the microprogram and hardware/firmware/software trade-offs will be studied.

The choice of intermediate language will be crucial to the design of this pipeline system. Trade-offs in processing times of different stages

Stage	Tasks Performed in the Stage
Input Pass:	Reads the source file line by line and reformats it so that the next stage needs a minimal amount of lookahead. It also computes hash functions for identifiers.
Lexical Analysis:	Creates symbol table and converts the character stream into a token stream.
Syntactic Analysis I:	Generates intermediate code for declarations, checks syntax and makes appropriate entries in the symbol table.
Syntactic Analyzer II:	Generates intermediate code for statements excluding expressions and assignment statement. Corrects syntax errors as best it can.
Expression Analyzer:	Handles expressions and assignments, and generates intermediate postfix.
Output Pass:	Generates machine code.

Fig. 5 The Functions Performed by the Proposed Six Stages of the PASCAL Pipeline Compiling Subsystem

offered by varying the intermediate language will be studied. In its preliminary design, the intermediate language described in Appendix B will be used. Since it is very close to machine language stage 6 should run fast.

12.3.3 Language Design

The language PASCAL has been modified for better handling of character strings and array operations and to implement some multiprocessing operators (see, for example [17]). Since the code generation consists of mapping code for a PASCAL machine onto a real machine, and since the mapping is done by a separate module, it should be relatively simple to generate code for a class of machines such as PDP-11's and Lockheed SUE's. Various language extensions are under consideration to control the creation and termination of dependent processes.

12.3.4 Analysis and Evaluation of Allocation and Schedule Algorithms

The problem of efficient processor utilization and resource management to achieve given measures of system efficiency and service throughput has been of both practical and theoretical interest in recent years. In a network containing a large number of processors and possibly a distributed data base, efficient resource management and scheduling algorithms become even more crucial to satisfactory overall system design.

As discussed in [18], we have studied several deadlock detection and prevention algorithms to determine their performance in a large network environment where control is distributed. We have studied those algorithms whose maximum execution time increases only linearly with the number of processes and resources in the system. Thus, the overhead of deadlock detection or prevention should represent a fixed percentage of the system

capacity. The objective of this study is to determine the relationship between maximum execution time and storage requirements for each of these algorithms. Whether or not the computation steps in these algorithms can be carried out concurrently and whether the various tables can be distributed in the system will also be investigated. We will modify these algorithms where possible to handle the case where the number of processes and the number of resources in the system may vary.

These same constraints lead us to investigation of several suboptimal job scheduling algorithms. In particular, we are concerned with a class of non-preemptive scheduling algorithms that never leave a processor idle. That is, when a processor becomes free, a task that is ready to be executed at that moment will be scheduled.[†] We have evaluated the worst case performance of these algorithms for multiprocessor systems containing processors of different speeds. The paper in Appendix D summarizes the results we have obtained to date.

The bounds on worst case performance of these scheduling algorithms also offer some insight in multiprocessor system design. For example, when the speed of the fastest processors is large enough,[‡] the maximum completion time of any set of tasks can be shortened by using the fastest processors alone.

[†] These algorithms are known as priority-driven scheduling algorithms.

[‡] By a factor of slightly larger than 1 to a factor of 5 or 6 as shown in Fig. 3 in Appendix E.

References

- [1] Liu, J. W. S. and G. L. Chesson, "Computer Networks - A Survey," Technical Report UIUCDCS-R-74-618, University of Illinois, Urbana, Illinois.
- [2] Davis, R. L., S. Zucker and C. M. Campbell, "A Building Block Approach to Multiprocessing," Proc. of AFIPS Conference, Vol. 40, 1972, pp. 685-703.
- [3] Baskin, H. B., B. R. Borgerson and R. Roberts, "PRIME - An Architecture of Terminal Oriented Systems," Proc. of AFIPS Conference, Vol. 40, 1972, pp. 431-437.
- [4] Wulf, W. A. and C. G. Bell, "C.mmp - A Multi-mini-processor," Proc. of Fall Joint Computer Conference, 1972, pp. 765-777.
- [5] Farbar, D. J., et. al., "The Distributed System," Digest of Papers, Compcon., 1973, Seventh Annual IEEE Computer Society International Conference, pp. 31-34.
- [6] Jasper, D. P., "A Definition of Networks," Digest of Papers, Compcon., 1973, Seventh Annual IEEE Computer Society International Conference, pp. 67-69.
- [7] Hassett, F. C., "Networks for Computer Utilities," Digest of Papers, Comp. Con., 1973, Seventh Annual IEEE Computer Society International Conference, pp. 195-198.
- [8] Thurber, K. J., E. D. Jensen and L. A. Jack, "A Systematic Approach to the Design of Digital Bussing Structures," Proc. of Fall Joint Computer Conference, 1972, pp. 719-740.
- [9] Miller, E. F., "Process Control and File Management Problems in Large Minicomputer Networks," Digest of Papers, Compcon., 1973, Seventh Annual IEEE Computer Society International Conference, pp. 199-201.
- [10] Kriloff, H. Z., "A High-Level Language for use with Multi-computer Networks," Proc. of National Computer Conference, 1972, pp. 149-153.
- [11] Gillies, D. B. and J. E. Vander Mey, "A Proposal for Research in Centralized Networks Employing a Diversity of Computing Elements," 1972.
- [12] Krabbe, P. "A Writable Control Store for Lockheed SUE 1110 Processor," Technical Report UIUCDCS-F-74-878, Department of Computer Science, University of Illinois, Urbana, Illinois.

- [13] Hart, J., "Emulation of the PDP-11 Instruction Set on the Lockheed SUE Processor," Technical Report UIUCDCS-R-74-621, Department of Computer Science, University of Illinois, Urbana, Illinois.
- [14] Davidson, S., "An Emulator for the LGP-21 Computer," Technical Report UIUCDCS-R-74-620, Department of Computer Science, University of Illinois, Urbana, Illinois.
- [15] Tao, W. Y., "A Firmware Data Compression Unit," Technical Report UIUCDCS-R-74-617, January 1974, Department of Computer Science, University of Illinois, Urbana, Illinois.
- [16] McKeeman, W. M., "Compiler Structure," Technical Report CSRG-23, January 1973, Computer Systems Research Group, University of Toronto, Ontario, Canada.
- [17] Hoare, C. A. R., "Hints on Programming Language Design," SIGACT/SIGPLAN Symposium on Principles of Programming Languages, Boston, Massachusetts, October 1-3, 1973.
- [18] Miller, T. J., "Algorithms for Deadlock Detection and Prevention," Technical Report UIUCDCS-R-74-619, Department of Computer Science, University of Illinois, Urbana, Illinois.

13. GENERAL DEPARTMENT INFORMATION

13.1 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	26	2	27.00
Visiting Faculty	3	1	3.67
Research Associates and Instructors	0	0	-----
Graduate Research Assistants	0	57	26.78
Graduate Teaching Assistants	0	34	17.00
Hourly Junior Academic	0	23	3.67
Professional Personnel	1	0	1.00
Administrative and Clerical	15	0	15.00
Nonacademic Personnel (Monthly)	12	1	12.50
Miscellaneous Supporting Staff (Hourly)	<u>0</u>	<u>8</u>	<u>2.45</u>
TOTAL.....	57	126	109.07

The Department Advisory Committee consists of Professor J. N. Snyder, Head of Department, Professors E. K. Bowdon, D. F. Cudia, M. F. Faiman, H. G. Friedman, C. W. Gear, D. B. Gillies, W. J. Hansen, W. J. Kubitz, D. J. Kuck, C. L. Liu, R. S. Michalski, M. D. Mickunas, R. G. Montanelli, S. Muroga, T. A. Murrell, J. Nievergelt, J. R. Phillips, 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.

13.2 Bibliography

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

Report Numbers

No.	598	Gear, C. W., "Asymptotic Estimation of Errors and Derivatives for the Numerical Solution of Ordinary Differential Equations," October, 1973.
No.	606	Bowdon, Edward K., Sr., "A Summary of Current Research in Computer Nets," November, 1973.
No.	607	Hansen, Wilfred J., "A Revised ALGOL 68 Hardware Representation for ISO-code and EBCDIC," November, 1973.
No.	610	Sameh, A., Lermitt, J., Noh, K., "On the Intermediate Eigenvalues of Symmetric Sparse Matrices," October, 1973.
No.	611	Schreiner, Axel T., "calc2 - A PLATO IV Lesson on Differentiation," December, 1973.

Theses

No.	582	Lerner, Edward M., "On-Line Filing (OLF) A Program Package for Student Records," October, 1973. (M.S. Thesis).
No.	583	Panigrahi, G., "Penetron Land Color Display System (PENTECOST) and Some Observations Concerning Color Perception," October, 1973. (Ph.D. Thesis).
No.	584	Sand, Douglas S., "COLFTAR: A Real-Time Electro-Optical System for Two-Dimensional Fourier Transforms," December, 1973. (Ph.D. Thesis).
No.	592	Donovan, Walt, "Design of an Algorithm for the Display of Visible Surfaces," October, 1973. (M.S. Thesis).
No.	594	Jauregui-Topete, Luis H., "Arrays in PL/I," October, 1973. (M.S. Thesis).
No.	595	Loessel, Mark C., "Design of a High Noise Immunity Analog Frequency Counter," October, 1973. (M.S. Thesis).
No.	596	Baker, Albert C., Jr., "A Generalized Lexical Scanner for a Translator Writing System," October, 1973. (M.S. Thesis).

Theses Cont'd

- No. 597 Edwards, Carol S. A., "Subgroups of the Group G_n ," October, 1973. (Ph.D. Thesis).
- No. 603 Weaver, Alfred C., "VIPTRAN - A Programming Language and Its Compiler for Boolean Systems or Process Control Equations," November, 1973. (M.S. Thesis).
- No. 604 Masumi, Ahmad E., "Picture Analysis by Graph Transformation," October, 1973. (Ph.D. Thesis).
- No. 612 Larson, John L., "Data Structures and Operator for New Array Types on OL/2," December, 1973. (M.S. Thesis).
- No. 613 Purvy, Robert E., "A Flowchart-To-Source-Code Conversion Program for GPSS/360," December, 1973. (M.S. Thesis).
- No. 614 Mueller, David H., "A Batch System with Rapid Overlay Capabilities," December, 1973. (M.S. Thesis).
- No. 616 Haskitt, Michael J., "A Two Compartmental Model of the Respiratory System," December, 1973. (M.S. Thesis).

13.3 Abstracts

Baker, Albert C., Jr., "A Generalized Lexical Scanner for a Translator Writing System," M. S. thesis, University of Illinois at Urbana-Champaign, Department of Computer Science Report No. 596, October 1973.

Abstract:

This is an expository paper that is concerned with a Lexical Scanner for a translator writing system that has been in use at the University of Illinois. Its significant features include a structured, binary-tree symbol table, a parameterized macro expander, and a compile-time flexibility for assigning characters that make up the basic terminal symbols. A comprehensive example of the scanner's operation is also included.

Bowdon, Edward K., Sr., "A Summary of Current Research in Computer Nets," Department of Computer Science Report No. 606, University of Illinois at Urbana-Champaign, November 1973.

Abstract:

In this paper we present some recent developments in the research program being conducted at the University of Illinois to formulate analytical tools for system modeling and analysis of real time computer networks. Specifically, we are conducting theoretical studies of geographically distributed network computers in three separate and distinct areas: Computer Network Modeling, Center Throughput Analysis, and System Performance Evaluation. Taken together these three areas provide the computer systems analyst with the necessary tools for modeling, analyzing, and evaluating system performance in existing and proposed network configurations - a first step in the long road to achieving economic viability in network computers.

Donovan, Walt, "Design of An Algorithm for the Display of Visible Surfaces," Department of Computer Science Report No. 592, University of Illinois at Urbana-Champaign, M. S. thesis, October 1973.

Abstract:

A visible surface algorithm capable of showing on a raster display smooth, transparent objects without jagged edges is described. The polygons defining the object to be displayed are put into visible order, and then output to the raster frame buffer by a special face drawing and shading algorithm. Very large objects or collections of objects can be manually put into visible order and each group drawn separately; this allows scenes with more faces than can be held in the program data structure at any one time. Minimum hardware for implementation is a raster display with a frame buffer, and a minicomputer with a disk.

Edwards, Carol S. A., "Subgroups of the Group G_n ," Department of Computer Science Report No. 597, University of Illinois at Urbana-Champaign, October 1973.

Abstract:

In this thesis the group, G_n , of permutations and complementations of n independent variables is embedded isomorphically in the symmetric group on the 2^n minterms of n variables, S_{2^n} . These minterms are represented in decimal notation.

For a subgroup H of G_n , the orbits of H play a key role in obtaining the major results. Necessary and sufficient conditions for an element of S_{2^n} to belong to G_n are also essential.

A subgroup H of G_n is said to fix a set of n -variable Boolean functions if every member of the subgroup leaves each function in the set invariant. For an arbitrary subgroup H of G_n , the largest subgroup M_H which fixes the set of functions fixed by H is determined, and an algorithm is given to compute M_H .

Algorithms are also given for the computation of N_H , the largest subgroup of G_n which leaves setwise invariant the set of functions fixed by H , and of $NG_n(H)$, the normalizer of H in G_n . Associated lattice structures are also studied.

Gear, C. W., "Asymptotic Estimation of Errors and Derivatives for the Numerical Solution of Ordinary Differential Equations," Department of Computer Science Report No. 598, University of Illinois at Urbana-Champaign, October 1973.

Abstract:

Shampine points out that the Milne device for estimating errors may give asymptotically incorrect results. This paper examines general classes of error estimates and shows how they can be used to form asymptotically correct estimates of the local error in the numerical solution of ordinary differential equations, or to estimate the derivatives of the solution of the differential equation.

Hansen, Wilfred J., "A Revised ALGOL 68 Hardware Representation for ISO-code and EBCDIC," Department of Computer Science Report No. 607, University of Illinois at Urbana-Champaign, November 1973.

Abstract:

Because of the latitude allowed by the Revised ALGOL 68 Report, each implementation has a slightly different representation for the constructs of the language. This diversity can only lead to confusion as ALGOL 68 trained individuals find they need readaptation

to program at a new installation. The solution proposed here is to develop a single hardware representation which can be used on many computer systems. In fact this representation can conveniently be designed using only the intersection of the graphic characters available in the ISO code and EBCDIC.

The paper also proposes comfortable new representations for a few symbols and discusses the thorny problem of distinguishing bold face words.

Haskitt, Paul R., "A Two Compartmental Model of the Respiratory System," Department of Computer Science Report No. 616, University of Illinois at Urbana-Champaign, M.S. thesis, December 1973.

Abstract:

A simplified two compartmental model of the human respiratory system is presented. The model is used to study the steady state responses to hypoxia at altitude and sea level, carbon dioxide inhalation, and metabolic disturbances, and the transient response to metabolic disturbances.

Jauregui-Topete, Luis H., "Arrays in PL/I a Lesson for PLATO IV," Department of Computer Science Report No. 594, University of Illinois at Urbana-Champaign, M.S. thesis, October 1973.

Abstract:

This is a report on the lesson, "ARRAYS IN PL/I", taught by means of the computer-based educational system, PLATO IV, to students taking introductory computer science courses. It presents the goals and decision procedures in writing the lesson, as well as the results of a trial with students.

Kuck, D., Budnik, P., Chen, S-C., Davis, E., Jr., Han, J., Kraska, P., Lawrie, D., Muraoka, Y., Strebendt, R., and Towle, R., "Measurements of Parallelism in Ordinary FORTRAN Programs," Sagamore Computer Conference on Parallel Processing, Syracuse University, pp. 23-29, August, 1973.

Abstract:

This paper reports the results of a measurement of parallelism at the statement level in 86 FORTRAN programs. The amount of parallelism is determined by an analyzer program and is measured in terms of speedup over serial execution, the number of independent processors required, the efficiency of parallel execution and other measures.

The analysis techniques are only sketched in this paper, details may be found in the reference. We also outline some machine organization assumptions.

Kuck, D., "Multioperation Machine Processing of Ordinary Programs," invited paper, to appear in First International Symposium on Computers and Chinese I/O Systems, Taipei, Taiwan, Aug. 1973.

Abstract:

This paper first outlines the history of high performance computer systems. Then a survey is given of upper bounds on the time required to evaluate various high level programming language constructs. Bounds on the number of processors required are also given. A number of details are given concerning high performance processing, aligning and accessing of data for parallel computation.

Larson, John L., "Data Structures and Operator for New Array Types in OL/2," Department of Computer Science Report No. 612, University of Illinois at Urbana-Champaign, M.S. thesis, December 1973.

Abstract:

This report is concerned with the addition of new array types to the OL/2 language. The necessary information for the implementation of Hessenburg arrays and their subarrays is given, including ACB representation and algorithms for partitioning. Modified data structures are proposed for describing band arrays and their subarrays, and new algorithms for partitioning are derived. Additional array types are realized by means of a rotate operator, which can generate new array types from the existing ones. Two possible implementations are suggested.

Lawrie, D., "Memory Systems for Parallel Array Processors," Proceedings of the Eleventh Annual Allerton Conference on Circuit and System Theory, Allerton Hse., Monticello, Ill., pp. 568-576, October 1973.

Abstract:

This paper discusses the organization of memory systems for parallel array processors which allow conflict-free access to various slices of data and subsequent unscrambling of the data to place it in correct order for processing. The discussion centers on conflict-free data storage techniques and data alignment networks.

Lerner, Edward M., "Øn-Line Filing (ØLF) A Program Package for Student Records," Department of Computer Science Report No. 582, University of Illinois at Urbana-Champaign, M.S. thesis, October 1973.

Abstract:

ØLF is a set of FØRTRAN programs which automates the maintenance and use of student records. Processing of information may be done in either batch or interactive mode. ØLF is designed to

Loessel, Mark C., "Design of a High Noise Immunity Analog Frequency Counter," Department of Computer Science Report No. 595, University of Illinois at Urbana-Champaign, M.S. thesis, October 1973.

Abstract:

This thesis discusses the design of a unique analog frequency counter. Frequency response is not as high as some counters currently on the market, being only 0 to 100 kilohertz. However, the technique used to convert analog input frequency to digital pulses provides more immunity to false triggering than any popular counter of today. The low-cost counter uses state-of-the-art components.

Masumi, Ahmad E., "Picture Analysis by Graph Transformation," Department of Computer Science Report No. 604, University of Illinois at Urbana-Champaign, Ph.D. thesis, October 1973.

Abstract:

In this thesis, we have developed a methodology to analyze the pictorial information represented in a map-like image, and further name the objects most likely present in the scene based on the universe known to the system.

In contrast to conventional string language sentences which are formed by concatenating the primitives (symbols) in a string according to the language grammar, the pictorial sentences are basically formed of primitives which are located in a two-dimensional (or three-dimensional) space with much richer relational properties than simple adjacency. Here we have examined the possibility of describing the structural information of pictures as syntactical rules, employing a rich class of relations. The graphs are known to be extremely convenient and flexible to represent this kind of information. By expressing the syntactical properties of one class of pictures as a collection of graphs, we have shown that algorithms can be easily developed to parse objects or collections of objects which belong to this class. Further, this methodology is flexible enough to recognize incomplete objects, and has learning capabilities through adding and modifying the rules of the system. In addition, we have shown that a complete description of an object as a collection of graphs can be easily modified (rotated) to enable the system to recognize different projections of the object. It has also been encouraging to note the heuristical information is easily introducible to the system and immensely improves the performance of the system.

Panigrahi, G., "Penetron Land Color Display System (PENTECOST) and Some Observations Concerning Color Perception," Department of Computer Science Report No. 583, University of Illinois at Urbana-Champaign, Ph.D. thesis, October 1973.

Abstract:

This work examines the human color vision mechanism in light of Dr. E. H. Land's two-color experiments on color vision. The problem of color perception in machines is looked into. Some of the two-color experiments are described. But the main emphasis has been on the building of a two-color television display system based on the two-color projection experiments.

Purvy, Robert E., "A Flowchart-To-Source-Code Conversion Program for GPSS/360," Department of Computer Science Report No. 613, University of Illinois at Urbana-Champaign, M.S. thesis, December 1973.

Abstract:

This paper describes an interactive program which allows a user to specify a GPSS program as a block diagram. The user draws the block diagram using the Graphical Remote Access Support System (GRASS) and sends it to the remote computer where it is converted to an equivalent source program. He can then run the program, if desired, and have the output from GPSS routed back to the graphics terminal, selecting the portion for which hard copy is desired.

The program is written in 360 Assembler Language.

Sand, Douglas S., "COLFTAR: A Real-Time Electro-Optical System for Two-Dimensional Fourier Transforms," Department of Computer Science Report No. 584, University of Illinois at Urbana-Champaign, Ph.D. thesis, December 1973.

Abstract:

An electro-optical system is described which generates optical Fourier transforms of two-dimensional video images, in real time, at standard video frame rates. This system is designated COLFTAR, which denotes "Cooled On-Line Fourier Transform and Reconstruction." The theory, construction, and operation of the system and its principal components are described in detail, with special emphasis on the resolution and modulation characteristics of the cooled-crystal electro-optic light valve (or Ardenne tube) which is the central component of the system. Several application examples are included, as well as a summary of possible improvements.

The system can produce Fourier transforms of arbitrary video frames with resolution of 500 by 500 line-pairs. Further, the same system can generate inverse Fourier transforms, so that an image can be reconstructed after its Fourier transform has been altered by an electronic "spatial filter." Also,

Fraunhoffer holograms and synthetic (computer generated) Fourier transforms be reconstructed. In operation the system can serve as a stand-alone device, as a computer peripheral unit, or-- using the advantage of real-time operation--for interactive image processing by Fourier transforms.

Schreiner, Axel T., "calc2 - A PLATO IV Lesson on Differentiation," Department of Computer Science Report No. 611, University of Illinois at Urbana-Champaign, December 1973.

Abstract:

calc2 is a lesson differentiation operating on the PLATO IV computer-aided instruction system. A symbolic differentiator guides the student through all steps of the differentiation of a student-chosen formula. The report describes the details of the table-driven symbolic differentiator.

Weaver, Alfred C., "VIPTRAN - A Programming Language and Its Compiler for Boolean Systems or Process Control Equations," Department of Computer Science Report No. 603, University of Illinois at Urbana-Champaign, M.S. thesis, November 1973.

Abstract:

The solution of some industrial process control problems can be expressed as a system of Boolean algebra equations using variables representing inputs from and outputs to the real world. If this system is solved sequentially, cyclically, and in real time, the outputs may be used directly to control the state of real world devices. VIPTRAN is a high-level FORTRAN-like language which allows the programmer to express his Boolean system symbolically in logical (Boolean) equations. The VIPTRAN compiler reduces the input to machine code for the VIP process controller, and can plot the resulting system as a relay tree. The concept is extensible to other industrial controllers.

13.4 Colloquia

"Software Physics," by Professor M. Halstead, Department of Computer Science, Purdue University, West Lafayette, Indiana, October 10, 1973.

"Mechanizable Proofs about Parallel Processes," by Dr. J. M. Cadiou, IRIA, Domaine de Voluceau, 78-Rocquencourt, FRANCE, October 22, 1973.

"On Finding the i-th Largest of n Elements," by Professor Frances Yao, Department of Computer Science, University of Illinois, Urbana, Illinois, October 31, 1973.

"Computer Controlled Switching," by Mr. D. E. Kaliher, Collins Radio Company, Cedar Rapids, Iowa, November 5, 1973.

"Pattern Recognition in Complex Environment," by Dr. K. Hanakata, Institut fur Informatik, Universitat Stuttgart, 7 Stuttgart 80, Pfaffenwaldring 64, West Germany, November 6, 1973.

"The Application of Computer Technology to Health Related Areas," by Dr. Ben Williams, Regional Health Resource Center, Urbana, Illinois, November 12, 1973.

"On Hypergeometric Group Testing Procedures," by Dr. Shen Lin, Bell Laboratories, Murray Hill, New Jersey, November 19, 1973.

13.5 Drafting

During the fourth quarter, a total of 460 drawings were processed by the general departmental drafting section:

<u>Formal Drawings</u>	
Large Drawings	25
Medium Drawings	117
Small Drawings	180
Layout Drawings	28
Report Drawings	57
Change Orders Drawings	29
Miscellaneous Drawings	24
Completed Total Drawings	460

(M. Goebel)

13.6 Shop's Production

Job orders processed and completed during the fourth quarter of 1973 are as follows:

	<u>AEC 2118</u>	<u>AEC 1469</u>	<u>Other</u>
Electronics Shop	1	36	31
Chemical Shop	1	20	12
Layout/Photo Shop	1	26	18

(F. P. Serio)