MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

GRAPHICS

SEMIANNUAL TECHNICAL SUMMARY REPORT
TO THE
ADVANCED RESEARCH PROJECTS AGENCY

1 JUNE – 30 NOVEMBER 1970

ISSUED 31 DECEMBER 1970

LEXINGTON MASSACHUSETTS

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by the Advanced Research Projects Agency of the Department of Defense under Air Force Contract AF 19(628)-5167 (ARPA Order 691).

SUMMARY

System architecture for the Terminal Support Processor (TSP) system is now complete. The principal element in the system is a dual processor version of the Digital Scientific Corporation Meta 4 microprogrammed computer. Main memory (magnetic core) consists of 8 banks of 8K 16-bit words. Secondary memory is provided by a one-million-word Digital Development Corporation disk having a 495-kHz transfer rate and an average access time of 8.7 msec. The TSP is connected via an Interface Message Processor (IMP) to the ARPA computer network and other Lincoln Laboratory computers. Consoles consist of a keyboard, a small number of visual and audible indicators, a graphic input tablet, and a pair of storage scopes. Displays are driven from a shared vector and character generator which is now complete and being tested. An extensive character set include most of the commonly used sets. A low cost input tablet has been developed utilizing a resistive sheet as the writing surface and an ordinary pencil as a pickup stylus.

TSP system software will be written in an extension of BCPL. BCPL programs will be compiled into code for a 16-bit machine called BCOM which will be simulated by microcode in the Meta 4 processors. The BCOM machine has an unusual addressing scheme oriented toward the addressing modes most often used by the BCPL compiler. Much IO programming in the TSP will be handled directly in microcode, and the processors in the system have been assigned specific system tasks to take advantage of the speed potential of specialized microcode. A simulator and assembler for Meta 4 microcode are now available on TX-2.

An IMP for connecting TX-2, the IBM 360/67, and the TSP to the ARPA network was delivered in July 1970, but connection to the network awaits installation of telephone equipment now scheduled for December 1970. The TX-2 interface hardware is connected and operating. The 360/67 interface is scheduled for delivery in December. Network control programs for both machines are under development.

Graph-theoretical work has yielded an improved planarity test algorithm. Its advantages and utility will be tested in close interaction with the Laboratory's integrated circuit design program.

Accepted for the Air Force
Joseph R. Waterman, Lt. Col., USAF
Chief, Lincoln Laboratory Project Office

CONTENTS

GLOSSARY

APEX	The TX-2 time-sharing system
BCOM	BCPL Compilation Oriented Machine – the simulated machine for executing TSP software
BCPL	Basic Combined Programming Language – an intermediate-level language for computer programming
CMS	The monitor system most widely used by on-line users of Lincoln Laboratory's IBM 360/67 computer
CP	Control Program – a time-sharing system for the IBM 360/67 computer
LIL	Local Interaction Language – a language for use in the TSP system
NCP	Network Control Program
RAP	ROM Assembly Program – an assembler for Meta 4 microcode
TSP	Terminal Support Processor

GRAPHICS

I. TERMINAL SUPPORT PROCESSOR (TSP) SYSTEM

The TSP system is intended to provide local support services to interactive graphics users of a computer network. The design is aimed at serving a maximum of 20 consoles, each consisting of a keyboard, a tablet, and a pair of storage scopes. Basic interactive graphics services include the following:

Keyboard echoing to displays
Tablet tracking and inking
Character recognition
Scaling and scissoring of picture information.

A major motivation for the TSP system design was the desire to provide effective interactive graphics services more economically than experience indicates they can be provided in a large general purpose computer environment. To accomplish this goal, the TSP design makes use of storage scopes to eliminate the large memory requirements of refreshed displays and time-shares a relatively powerful processor configuration.

The TSP has system programs to provide the basic IO services required by an interactive graphics user, but experience shows that users require greater flexibility than can be provided by any package of system services. To meet such needs, the system provides a language called LIL (Local Interaction Language). By writing programs in LIL, a user can control interaction between his console input and output devices and between his LIL program and other programs in the computer network. LIL is a general purpose language with primitives for manipulating display structures and handling message-oriented input-output.

The TSP system has the form of a small general purpose time-sharing system oriented toward effective handling of small programs requiring short bursts of execution and fast response. The TSP provides no file system, and user programs must be kept in some other computer in the network. Although the TSP and LIL offer a general purpose capability, users are encouraged to view the system as providing a sophisticated console rather than another computational facility.

Effective use of the TSP system assumes that a user can partition his overall problem between the TSP and some other computer (or computers) in the network. If the IO interactions can be handled directly in the TSP while large computations are being carried on elsewhere in the network, a satisfactory interactive environment may be presented to the user. It is probably true that some interactive applications exist which cannot be adequately partitioned, but analysis of existing interactive programs at Lincoln Laboratory suggests that many problems of interest and importance can be handled in this way.

The TSP is being developed as an independent host computer in the ARPA computer network. It will be connected to the Laboratory's IMP along with TX-2 and the IBM 360/67. The TSP will send and receive messages in a manner identical to other host computers, but it will not act as a serving host for users at other hosts.

The system architecture for the TSP is now complete. All system hardware has been ordered from vendors or is being constructed in the Laboratory. Delivery of all system components is expected during the first quarter of 1971. A description of the TSP architecture is presented in Secs. I-A through I-C with the status of each system component indicated.

A. TSP System Hardware

1. Processors and Memory

The principal element in the TSP is a dual processor version of the Digital Scientific Corporation Meta 4 computer (see Fig. 1). This is a high performance, microprogrammed, 16-bit machine with a typical microinstruction execution time of 90 nsec. Microinstructions are fetched only from a plug-in, read-only memory (ROM) which has a maximum capacity in each central processor (CP) of 4K 16-bit words, or 2K 32-bit microinstructions.

So-called "firmware" routines, in the ROM, access 16-bit words from main memory and manipulate them in general registers in the CP. Although up to 32 registers are available in the Meta 4, in the TSP configuration only 20 will be implemented in each CP. A typical microinstruction combines the contents of two of these registers and puts the result into a third. Most of these combinations realize simple Boolean and arithmetic (2's complement) functions. Some of these registers also help realize special functions, some of which relate to communications outside the CP.

The main memory utilizes magnetic cores and consists of 65K words arranged in 8 banks of 8K words each. Each bank has four ports, one for each of the two CI's and one for each of the four input-output processors (IOPs). The ports on each bank respond in a nonclocked manner to access requests, and are priority ranked in order to resolve simultaneous request conflicts.

The read-restore and clear-write main memory cycle time is 900 nsec. Hence, in order to maintain maximum CP utilization of main memory exactly 10 microinstructions should be executed for each memory access. In practice this 10:1 ratio need not be exactly maintained: The memory will always wait asynchronously for a request, and suitable interlocking is built into the CP request mechanism so that firmware routines will wait for memory to be available.

Each CP communicates with memory via two dedicated general registers in the CP. One is used to send the memory address and the other to send or receive the memory word. Special bits in microinstructions which reference these registers indicate whether a read or write operation is to be initiated.

Each CP also communicates with an associated IOP via another pair of special registers in the CP. These same special bits are used, in this case, to direct whether input-output (IO) operations are to be initiated and, once initiated, how to interlock the execution of microinstructions with them.

IO operations can be either direct, or on a cycle-stealing basis. In the direct case the firmware selects an IO device and causes a word transfer between it and the CP. The selection and transfer paths are via the two special registers. In the cycle-stealing case, word transfers occur directly between the device and the memory via the IOP and the port connection to memory. The direct mode is used to initiate the cycle-stealing transfers; no further involvement of the CP is required until some change of state in the device requires that an asynchronous pseudo-interrupt be sent to the CP. Approximately 20 cycle-stealing devices could be operating simultaneously in an interleaved fashion on each CP. Although a similar number of direct IO devices could also be running simultaneously, a CP could communicate with only one at a time.

The current schedule calls for the Meta 4 computer hardware to be delivered to Lincoln Laboratory in January 1971.

2. IO Connections

Figure 2 shows the principal IO devices to be connected into the TSP system. The following paragraphs discuss each device in turn, indicating its function in the system and its present delivery or development status.

a. IMP

The IMP connection provides access to the ARPA network, and, in addition, it will provide the initial means of loading programs into the TSP system from either of the two other host computers at Lincoln Laboratory (TX-2 and the IBM 360/67). A full duplex, cycle-stealing connection can allow information transfers in each direction at a bit rate well in excess of the standard IMP/HOST rate of approximately 100 kHz.

The IMP itself was delivered to the Laboratory in July 1970. The interface hardware is being built in the Laboratory and is expected to be ready for testing when the Meta 4 computer is delivered and accepted, so that it can be used for initial checkout of the TSP system.

b. Secondary Storage

Within the TSP system, secondary storage is provided by a 1,000,000-word Digital Development Corporation disk. This will have a word transfer rate of 495 kHz and an average access time of 8.7 msec. Good performance of the TSP system requires that user programs and data be rolled on and off the disk with high speed and efficiency. The disk controller achieves this by providing both command and data chaining capability, thereby minimizing main memory moves and/or wasted disk space and access time. All command and data word transfers use cycle-stealing access to memory. The disk controller is being provided by Digital Scientific Corporation with delivery expected during the first quarter of 1971.

c. User Console

All the user consoles together comprise three distinct IO devices: the storage scope displays, the tablets, and the keyboards and indicators. Each of these devices has its own multiplexing scheme tailored to the requirements of the device. All are being developed in the Laboratory. Section I-D discusses further the console hardware.

Keyboards and Indicators:- This will be a low data rate, directly programmed (non-cycle-stealing) device which sends information to the CP whenever a user presses a key or button and receives information which causes lights or other indicators to turn ON or OFF. A time-multiplexed, serial transmission scheme is being explored for this device. Every piece of information will be in a word which contains a console identification tag.

Tablets:- Each of the tablets must be sampled for two words of data between 150 and 200 times per second. Since the system must process each tablet reading before the next arrives from the same tablet, a direct IO connection would appear natural, but the interrupt overhead in this mode would be unnecessarily high. By using a cycle-stealing connection to enter the tablet data into buffer blocks in memory, a single interrupt per scan of all tablets is sufficient. The design will avoid cycle-stealing as well as program load for inactive tablets.

Displays:- The multiplexer which drives the storage scopes is the display generator itself. In the full TSP system there will be two display generators, each driving the storage scopes at ten consoles. The LX-1 microprocessor is interposed between the Meta 4 and the display generators to provide scaling, scissoring and other services.

LX-1 acts as a cycle-stealing IO device to the Meta 4 system, but this connection operates as a memory port expander so that LX-1 can access memory in a manner similar to a CP. LX-1 is a microprogrammed computer, also, but its instruction memory is not a ROM. This memory needs to be loaded at system load time; hence, a second IO device connection will be built to realize this operation. This will be a cycle-stealing device which will be under the control of the Meta 4.

d. Miscellaneous IO Devices

The operator's console, a real-time clock, program stall and power fail alarms are standard Meta 4 IO devices which are included in the system. In addition, the Laboratory will build a program controlled interrupt device to facilitate interprocessor communication and a special device to capture information about memory parity errors and protection violations occurring in the IOP's.

B. TSP Console Hardware

1. TSP Displays

The TSP display generator which consists of a vector generator and a character generator has been constructed and connected to our SEL test computer for evaluation and test. The SEL computer simulates both the LX-1 from which the display generator will be driven, and the ROM for the character generator, which is not yet available. With this connection the full capability of the display generator can be exercised, and the ROM contents can be verified prior to construction. Test results are very encouraging.

The display generator has a 10-bit resolution and uses the multiplying D-to-A converter (DACs) developed for the TX-2 conic generators for its operation. However, these multiplying DACs are used in a different manner. In the TX-2 conic generator, a stroke or vector is generated by a pair of parametric equations

$$\begin{aligned} X &= X_o + \dot{X}t \\ Y &= Y_o + \dot{Y}t \end{aligned} \qquad t\Big|_o^T$$

where X_o, Y_o, $\dot{X}$, $\dot{Y}$, and t are the starting positions, velocities and total drawing time, and are explicitly given. A diagram of the vector generation portion of the TX-2 conic generator is shown in Fig. 3.

In the TSP display generator a vector is produced by the following operation:

$$\begin{aligned} X &= X_o(1 - V) + X_1 V \\ Y &= Y_o(1 - V) + Y_1 V \end{aligned} \qquad V\Big|_o^1$$

where X_o, Y_o, X_1, Y_1 are the starting and the ending coordinates of the vector and V is the reference to the multiplying DACs. The hardware realization of this scheme is shown in Fig. 4. This method of vector generation is conceptually simpler. It simplifies the programming for

drawing connected lines because only successive end points need be given. In addition, the hardware requirement is also reduced.

One implicit parameter in the TSP scheme is time, which is contained in V. In order to draw a vector at a constant speed, the time for $V|_0^1$ must be proportional to the vector length. Otherwise, intensity variation and variation of the line width will result. Experiments show that a 25-percent speed variation is acceptable, and the TSP display generator is designed to keep the speed variation less than that value. This is accomplished by varying both the number of steps in the full count and the rate to which the steps are added.

The character generator operates on the same principle, except that the range is only 5 bits and the character stroke data are provided from a read-only memory. Two mode features, the slanting of characters and a 90° rotation of the characters, are provided by the character generator.

The character set for the TSP system is shown in Chart I. The design attempts to provide a comprehensive set that will satisfy the needs of most users. The set includes the characters of the ASCII, IBM 360 system, Teletype, APL\360* and TX-2 sets as well as some additional mathematical and phonetic symbols. The chart can be generally divided into two halves. The left half of the chart contains the ASCII characters. The right half contains the extra characters needed to make up the following sets:

(a) APL, mathematical symbols and upper case Greek letters, (columns 1010, 1011, 1100).

(b) Phonetic symbols (column 1101).

(c) Lower case Greek letters, special symbols, a nondisplaying dummy and a symbol for all undefined character codes which happens to be the Chinese character for forest (11111110).

Samples of the character generator outputs are shown in Figs. 5, 6, 7.

2. TSP Tablet

A low cost input tablet for the TSP was fabricated and tested. It uses a resistive sheet as the writing surface and an ordinary pencil as the pickup stylus. It is a potentiometer device and depends on direct contact to function. A diagram of the tablet is shown in Fig. 8.

A low frequency sinusoidal voltage is applied to a conductive sheet through four sets of diodes, setting up a potential gradient along one axis of the tablet during one half-cycle of the sine wave and along the other axis during the other half-cycle. The gradient is fixed by the material distribution of the resistive sheet, although the magnitude is time varying. At any instant the voltage at any point on the tablet is directly related to the distance of that point from the driven edges of the tablet.

In order to calibrate, or normalize, the tablet to some standard scale, a normalization probe for each axis is placed at the tablet's edge just beyond the normal writing area. The outputs of these probes are fed to an analog comparator. When the output of a normalization probe reaches a reference voltage, the stylus pickup voltage is sampled and converted. One conversion for the horizontal position is made during the first half-cycle of the driving sinusoid and one for the vertical position during the second half-cycle. Experiments using this tablet on TX-2

* APL\360, an IBM program product, is a conversational terminal system based on a language described by K. E. Iverson in A Programming Language (Wiley, New York, 1962).

$b_4 b_3 b_2 b_1$ \ $b_8 b_7 b_6 b_5$	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
0 0 0 0					Ø	@	P	`	p			¯	§	∴	æ	ə	ρ
0 0 0 1				!	1	A	Q	a	q			□	ſ	•	ɔ	ɑ	σ
0 0 1 0				"	2	B	R	b	r			▢	ι	¨	ʊ	β	τ
0 0 1 1				#	3	C	S	c	s			·	▽	Γ	ə	ɣ	ν
0 1 0 0				$	4	D	T	d	t			⊔	⊥	Δ	ɦ	δ	φ
0 1 0 1				%	5	E	U	e	u			∩	⊤	Θ	ð	ϵ	χ
0 1 1 0				&	6	F	V	f	v			⊃	∣	Λ	ʃ	ʔ	ψ
0 1 1 1				'	7	G	W	g	w			⊂	‖	Ξ	ʒ	ɳ	ω
1 0 0 0				(	8	H	X	h	x			∨	⫟	Π	ŋ	θ	∞
1 0 0 1				)	9	I	Y	i	y			∧	⫤	Σ		ι	¢
1 0 1 0				*	:	J	Z	j	z			≥	∃	T		κ	¤
1 0 1 1				+	;	K	[	k	{			≤	∀	Φ		λ	¬
1 1 0 0				,	<	L	\	l	¦			↓	≠	Ψ		μ	‿
1 1 0 1				-	=	M	]	m	}			↑	≡	Ω		ν	ˇ
1 1 1 0				.	>	N	^	n	~			→	×	∂		ξ	‡
1 1 1 1				/	?	O	_	o				←	÷	∫		π	NON-DISPLAY DUMMY

Chart 1. TSP character set/code correspondence.

showed that this staggered sampling does not produce any noticeable distortion of handwr[illegible]n characters, as long as the sampling frequency is above 100 Hz. For the final system, the operating frequency probably will be between 150 to 200 Hz.

The tablet sheets are made from tin-oxide coated glass plates normally used as fluorescent light diffusors and RFI shields. Driving contacts to the tablet's edges are made through ten spring-loaded pins. Since the sampling rate is low, a single analog-to-digital converter can serve many tablets, and since no expensive components are required, it is expected that a very low cost multitablet system will be achieved.

The experimental tablet was built and tested on TX-2 for a period of two months. It worked fairly well. There was no noticeable wear on the tablet surface. The resolution and linearity were adequate. One problem uncovered was noise in the pickup signal which occurs when the stylus is making or breaking contact with the tablet surface. Several schemes for avoiding the noise by detecting a poor contact condition have been tried, including RF impedance detector, DC level detector, and frequency discriminator. None worked well. Currently, a scheme using a strain gage is being investigated. In the final system, the Meta 4 firmware will do some simple filtering on the tablet data and hopefully minimize this noise problem.

3. Keyboards and Indicators

The TSP console will contain a keyboard and a small number of indicator lights and audible signaling devices. The keyboard design assumes a standard commercially available keyboard with, at most, a very few extra keys. Since the TSP character set is quite large, multiple shifts will be required to key all the characters from a standard keyboard. Several commercial keyboards having both mechanical and solid state encoding schemes have been purchased for evaluation. Several proposals for keyboard layout have been generated by representatives of the prospective user's community. The many implications of the hardware, firmware, software, and human factor aspects of the keyboard design problem are now being studied, and keyboard selection should take place in the very near future.

C. TSP Programming

The microprocessors in the TSP system are capable of executing simple program steps at a rate of 10 per main memory cycle. If all the necessary system programs could be contained in the read-only memories of the microprocessors, very high system performance would be achieved, but only about 4K words of such memory are available and extrapolation from the TX-2 system indicates that at least 24K words of program will be required to provide all the TSP services. It is therefore necessary to use the microprocessor to simulate a processor which can draw its instructions from the larger but slower core memory. Such a simulated processor will execute program steps more slowly, but programs can be much larger and can be much more readily changed. Preliminary investigation suggests that a simple 16-bit processor can be simulated in a Meta 4 microprocessor, using about half of the available ROM space. If the other half of the ROM is used to execute small routines directly in microcode, and if these routines are called frequently during normal operations, substantially higher overall performance can be achieved. The TSP system design makes use of this technique, realizing some system functions in "firmware" (ROM program executed directly by the microprocessors) and some in "software" (main memory programs executed by simulation). In the following discussions the terms "program" and "routine" will be used as general terms applying equally to firmware or software.

1. System Software

System software will be written in an extension of BCPL, an intermediate-level language reasonably well suited to the kind of programming needed for the system and familiar to a major segment of the programming group. BCPL compiles into a machine-independent intermediate form which allows the generation of a compiler for a new machine to be accomplished relatively easily. Some thought was given to interpreting this intermediate form directly in firmware, but it appeared that little would be gained in run-time efficiency, and that the flexibility of being able to write in assembly code would be lost. Instead, it was decided to define a convenient machine language into which BCPL will be compiled. The resulting 16-bit machine is called BCOM (for BCPL Compilation Oriented Machine) and will be simulated by firmware in the Meta 4 microprocessors. BCPL compilation will take place in TX-2, and BCOM binary will be transmitted to the TSP system via the IMP whenever a new system is to be generated. A BCOM assembler will also be available in TX-2 for maintenance, IO, and other programs not readily handled in BCPL.

2. User Software

User software will be written in LIL (Local Interaction Language). (See Graphics Semiannual Technical Summary for 31 May 1970 for a discussion of the user specifications of LIL.) Implementation plans call for LIL programs to be compiled in TX-2 or some other large machine in the network and transmitted to the TSP system in a form similar to machine language with the exception that all addresses of executable code are symbolic. This compiled form will be loaded into program segments by means of LIL primitives which will structure the program segments so that they may be readily modified at run time. Not all LIL programs need be self-modifying, but those representing commands to generate displays must be, if the resulting picture is to be modified in response to a user interaction. The actual bit patterns representing the structured program in TSP memory may be considered to be LIL machine language. Except for debugging the system, this machine language form, which is quite arbitrary since there is no corresponding real machine, need not be accessible to users or their programs.

The LIL machine will be simulated by a combination of firmware and software. Simple primitives, such as many in the general purpose parts of the language, can be handled directly in firmware, but more complex primitives, such as those for program structuring and IO control, must be handled with software routines because the ROM space for firmware is much too small. The software will not be realized in LIL machine language itself, because the LIL symbolic addressing scheme requires translation of all addresses, an unnecessary source of overhead in system routines. Instead, these LIL run-time routines will be realized in the same fashion as other system software (i.e., written in BCPL and executed in a simulated BCOM).

3. The BCOM Machine

The basic problem in designing an order code for a 16-bit machine is deciding how memory will be addressed. Since BCPL, the language for which the BCOM machine is being designed, is a stack-oriented language, an obvious first choice is to provide a hardware stack pointer and stack pointer relative addressing. However, not all addressing can be stack relative. Constants known at compile time and static variables will not be on the stack. Constants may be embedded in the executable code and accessed via program counter relative addressing. Static variables have addresses which are known at compile time. These addresses may be embedded in the

executable code, and the variables accessed via program counter relative addressing followed by an indirect cycle.

Since the value of any BCPL variable may itself be an address, it is convenient to add a level of indirect addressing to the addressing modes described above. It is also common (e.g., in vector application) to treat the sum of two variables as an address. This can be done conveniently by providing an index register in addition to the stack pointer. First, the value of one variable is loaded into the index register. Then an addressing mode is used which extracts the value of the second variable, adds the contents of the index register, and uses the result as the effective address.

The eight addressing modes of the BCOM machine are:

Effective Address	Used to Obtain
±Δ + PC	Value of a constant or address of a static variable
+Δ + SP	Value of a variable on the stack
+Δ + X	Value of a vector subscripted by a small constant
(±Δ + PC)	Value of a static variable
(+Δ + SP)	Value of a variable whose address is contained in a variable on the stack
(±Δ + PC) + X	Value of a vector subscripted by a large constant
(+Δ + SP) + X	Value of a vector on the stack subscripted by a variable, or of a static vector subscripted by a variable on the stack
((±Δ + PC)) + X	Value of a static vector subscripted by a static variable

where the following definitions hold:

Symbol	Meaning
+Δ	Contents of the address field of the instruction, interpreted as an 8-bit positive integer
±Δ	Contents of the address field of the instruction, interpreted as an 8-bit signed integer
PC	Contents of the program counter
SP	Contents of the stack pointer
X	Contents of the index register
(Z)	Contents of memory location Z

This addressing scheme may appear strange to assembly language programmers more familiar with hardware-oriented machine designs. Its justification is that these are the addressing modes most often used by the BCPL compiler, and therefore their use makes most efficient use of the three bits available for address-mode specification.

In other respects, the BCOM design has developed along relatively more conventional lines. A complete basic machine has been specified with room for extension to meet special demands and to allow further optimization as the compiler develops. Meta 4 firmwave to simulate BCOM has been written and is being debugged using the Meta 4 simulator on TX-2. It is expected that delivery of a ROM to simulate BCOM will match the delivery of the Meta 4 processors.

4. IO Programming

All IO devices in the TSP system require some firmware to support them. For functionally simple devices such as the disk and the IMP, firmware services will be limited to hardware acknowledgment of interrupts and queueing of interrupt events for software processing. For the functionally complex devices (the keyboards, indicators, tablets, and displays) firmware services will be more extensive. By handling simple, frequently exercised routines such as tablet tracking and inking, and display scaling and scissoring directly in firmware, advantage can be taken of the high execution speed of microcode. By similarly handling infrequent tasks requiring fast response, such as keyboard character echoing to displays, it is possible to avoid software level interrupts with consequent gain in efficiency and simplification of software design. Current plans call for all system software services to be called by a dispatcher working from a task queue. Software routines must run to completion or requeue themselves appropriately. Software response times are therefore anticipated to be on the order of milliseconds. Situations requiring faster response will be handled by firmware or IO channel command chaining.

5. Processor Specialization

In order to take advantage of the speed potential of specialized firmware routines, the three microprocessors in the TSP have been assigned specific system tasks. Figure 9 is a simplified representation of the system showing the principal IO paths, the three processors, and memory. The principal functions performed by the processors both in software and firmware are:

Processor A (Meta 4 CPA)

Firmware	BCOM simulation Tablet tracking and stroke demarcation Keyboard character handling and string demarcation
Software	Network control program (NCP) functions Scheduling Disk allocation Character recognition User IO stream management

Processor B (Meta 4 CPB)

Firmware	LIL simulation BCOM simulation
Software	LIL run-time routines

Processor C (LX-1)

Firmware	Translation of user display output streams Scaling and scissoring of picture information Display tablet tracking bugs and ink Echo keyed input characters
Software	None

It may be noted that both the Meta 4 processors (A and B) have firmware for BCOM simulation, and this arrangement will allow certain pieces of system software to be moved from one processor to the other to equalize system load, but there is no plan to treat the two as a general multiprocessor configuration.

D. Supporting Software

TSP implementation plans call for most supporting software to operate on TX-2 with system loading through the IMP connection. Compilers are needed for LIL and BCPL, assemblers for BCOM and for Meta 4 and LX-1 microcode, and simulators for the Meta 4 and LX-1 to allow firmware to be substantially debugged before being committed to actual read-only memory. TX-2 editors and some debugging facilities can be used directly, but some new utility and debugging programs are planned to allow TSP memory and machine states to be examined from TX-2 consoles. Work has not yet been started on the LIL and BCPL compilers. Work has just begun on a BCOM assembler, and it should be available when needed to prepare checkout programs for the hardware. The LX-1 assembler and simulator have been available for some time, but a few changes are necessary to reflect hardware design modifications. The status of the Meta 4 microcode assembler and simulator are discussed in the following two sections.

1. Meta 4 Simulator

A simulator for the Meta 4 computer has been written. It operates under the APEX time-sharing system on the TX-2 computer. The entire microinstruction set of the Meta 4 is implemented in the simulator with the exception of the two input-output (IO) instructions.

The simulator has been checked out by using it to run a series of small Meta 4 programs designed to test the simulation of the various microinstructions, instruction modifiers and combinations thereof. The simulator appears to operate the way we understand the Meta 4 to operate. One or two programs which are known to run on a real Meta 4 have been run under the simulator and have produced the correct results. This second phase of simulator checkout will be continued.

The simulator has not been programmed to handle the two IO instructions because many of the exact details of the IO configuration are not known. As this information becomes available, the simulator will be expanded to handle those IO devices which it appears it would be useful to simulate.

The primary use of the Meta 4 simulator will be to verify the system firmware before the actual read-only memory is fabricated. Consequently the simulator offers many debugging and checking features designed to alert the programmer to errors in the code he has written. When the simulator encounters a microinstruction which is in error, the instruction is not simulated. Instead a message is typed on the user's console. The user is told the location of the offending instruction and what is wrong with it. The simulator can catch such errors as: illegal general register usage, illegal core memory reference, illegal ROM reference. Timing checks are included in the simulator so that the user is told if his program violates the timing restrictions which exist in the current Meta 4 hardware when core memory is referenced.

The user is provided with the capability of monitoring his program as it is being run by the simulator. The user may specify that he wishes to see the results of his program's operation after it has run any given number of instructions. On a storage scope at the user's console, the simulator displays the contents of the 20 general registers in the Meta 4, marking those that have changed since the last display. The display also includes the program counter and instruction representation for the microinstruction just simulated, as well as the location of the next instruction to be simulated. When the user finishes looking at the display, he may hit a specific key on his typewriter to cause the next instructions to be simulated and the results displayed.

While he is looking at his display, the simulator is in an interrupted state and control has been passed to RAP, the microcode assembly program, allowing the symbolic debugging features being designed for RAP to be implemented easily.

2. RAP – A Meta 4 Microcode Assembler

An assembler called RAP for the ROM of the Meta 4 was produced and is now being used.

RAP accepts symbolic source language statements representing Meta 4 microinstructions and generates a binary TX-2 file which can be used as input to the Meta 4 simulator.

The source language format was derived from the manufacturer's assembler specifications with modifications to adapt the language to the TX-2 environment. With only one exception, names for instructions and modifiers were carried over directly. The format of an instruction line was changed drastically since the manufacturer's card column format is not compatible with the TX-2 environment which lacks card facilities. The new format allows variable size identifiers with the order determining the meaning. Numbers are accepted in octal as well as hex notation.

A working version of RAP was achieved in 2 man-days from initial conception by starting with Mark 5 (TX-2's assembler) and making appropriate modifications (e.g., reserved symbols, meaning of the various fields). As a result, much of the power and facilities of Mark 5 carry over directly. For example:

> RAP is an on-line interactive program in the APEX TX-2 time-sharing system environment. Hence one may make changes to a program and see the effects of the changes almost immediately. Many steps in the evolution of a program are possible in one console session.
>
> A macro facility is available.
>
> Editing of a program may be done via any one of 3 editors available in APEX.
>
> Listings may be obtained via the console typewriter, console display scope or high-speed xerographic printer.
>
> Extensive concordance facilities are available to enable one to find all occurrences of identifiers and to edit all occurrences at the same time.

A direct interface between RAP and the simulator is currently being designed in order to provide an integrated symbolic assembler-debugger for ROM programs.

II. ARPA NETWORK

Current plans for the ARPA network call for the connection of three Lincoln Laboratory computer systems into the network. These are: (1) The Laboratory's principal service facility, an IBM 360/67, (2) the TX-2, an experimental computer which has been supporting the Graphics program, and (3) the new TSP system discussed above. This section reports the status of work on the connections and the supporting software for the 360/67 and TX-2.

A. Network Hardware

The ARPA network IMP at Lincoln Laboratory is configured to handle two 50-kbit telephone channels and three host computers – one local host and two distant hosts. The local host machine will be the Lincoln Laboratory IBM system 360 Model 67 computer; one distant host machine will be the TX-2 computer and the other will be one of the CPUs in the TSP system.

The connection between TX-2 and the IMP consists of a full duplex, time-interleaved (simultaneous bidirectional) buffered data path. Although the TX-2 word size is 36 bits, the word size

transmitted over the TX-2 IO interface to and from the IMP is 32 bits (8 lower bits of each quarter word in TX-2). This unusual interfacing arrangement is intended to increase the efficiency of handling 8-bit byte-oriented transmissions which are expected to dominate network traffic. The data buffering in each direction consists of a full width buffer register plus a full width shift register. The data path for each direction is handled by a separate TX-2 IO Sequence. Both data paths are capable of operating over direct memory access cycle-stealing channels; however, the program is being written to cycle-steal only on the output TX-2-to-IMP data path due to the limited number of available channels. The input data path will utilize program interrupt IO facilities.

The IMP was delivered to the Laboratory in July 1970. Since that time the interface has been under test, "echoing" messages back and forth between the IMP and TX-2 over a temporary 25-foot cable. Integration of the IMP into the network awaits installation of the T-carrier telephone equipment which is now scheduled for early December. Installation of the permanent 1300-foot remote host cables is under way; the IMP will be moved to its final location and tested with the permanent cable prior to arrival of the telephone equipment.

The 360/67 interface has been ordered from the University of California at Santa Barbara and is scheduled for delivery in mid-December. It is a full-duplex connection to the 360 multiplexor channel.

B. 360/67 Network Software

CP-67 is the time-sharing system which runs on the Laboratory's IBM 360/67. It provides virtual System 360 machines for on-line users. In order to provide network access to a virtual machine, some modifications and functional extensions to the CP-67 system will be required. To minimize the effect on CP during the developmental stages and the initial use of the network, the IMP will be directly attached to a virtual machine and will be driven by code in the virtual machine rather than from code in the real-core CP system. This virtual machine will contain the Network Control Program (NCP) whose function is to handle network scheduling and protocol, and to control and account for network access to the CP system.

A rendezvous table will be used to schedule the virtual machine NCP which will be interrupt-driven through the following software interfaces:

(1) User's processes communicate with the NCP through NCP commands. For each command issued by a user, a reply is returned giving the status of the socket. In addition, the NCP generates an interrupt to a user process whenever:

- A send connection is completed, i.e., when an NCP control message is received from a remote NCP which matches a previous local request to establish a send connection
- A connection is closed by the remote process or host
- A message is received from a remote process and no message is currently stacked for the receiving process
- A remote process transmits an interrupt message.

(2) The IMP/OUT software interface will process requests to transmit information over the network. This routine issues the IO commands to the hardware interface to transmit to the IMP.

(3) The IMP/IN interface will receive messages from the network, initializing or modifying entries in the rendezvous table, transferring messages to the appropriate entry in the rendezvous table for output through a different interface. This routine issues the IO commands to the hardware interface to receive data from the IMP.

(4) The NCP software interface to CP and the virtual consoles associated with processes will be controlled by a logger routine which receives input from the IMP/IN interface and passes output to the IMP/OUT interface.

(5) Local terminals will also be controlled by the NCP, enabling local users to connect directly to the NCP and thus to the network without going through a CP virtual machine.

Through the terminal associated with the NCP virtual machine, network status can be monitored and special corrective action taken to insure the integrity of the rendezvous table.

The logger running in the NCP virtual machine will cause a new virtual machine to be created for a network user and will establish connections between NCP sockets and the virtual machine to simulate a local typewriter terminal. In this way, a remote network user can become a user of the Lincoln Laboratory CP-67 system to run any of the available virtual machine operating systems.

CMS is the monitor system most frequently run in CP virtual machines. A CMS user will be able to:

Initialize a connection with a process in some host computer on the network

Communicate with the remote process in a manner similar to a local user, i.e., similar to a user logged in on a local terminal

Send and receive a file or data set over the connection

Disable the connection.

A user process running under CMS calls on a program called NET to communicate with the NCP. At any time, the user process can make a call to a CMS service function to wait for an interrupt from the NCP. When the interrupt occurs, the code and status information can be obtained to indicate the nature of the interrupt.

In order for a CMS user to act as a user of a remote process, it will be necessary to modify the CMS system to operate the terminals in a manner supporting asynchronous input and output. Currently, the operation of a CMS terminal is controlled by the CMS system programs such that keyboard input is only permitted when requested by a program, and output does not interrupt keyboard input. Since the CMS system communicating with a remote process does not know when input is required and must be able to print output when it is received, CMS will be modified to permit keyboard input to be controlled by the user and will force terminal output when it is received.

The user interface to the NCP has already been implemented, and work is progressing on the IMP/IN and IMP/OUT interfaces which will be completed by the time the 360/IMP hardware interface is installed.

The logger interface should be operational in January 1971, as well as a number of user facilities for conversing over the network and for transmitting files over the network.

C. TX-2 Network Software

As part of an experiment to gain experience with alternate protocols for the ARPA network, an implementation of the protocol described by R. Kalin in "Network Working Group/Request for Comments," No. 60, was begun. This protocol differs from the official network protocol (Host-Host Protocol, Document No. 1, by S. Crocker) in two major ways:

(1) Buffer allocation is on a per-message basis, rather than on a per-bit basis.

(2) The NCP need allocate space only in response to local user requests, never in response to network messages.

An NCP implementing the Kalin protocol has been completed, but results of the comparison with the Document No. 1 protocol will not be available until both have been finished.

The TX-2 NCP is scheduled for completion by mid-January 1971. The necessary changes to other parts of the APEX system to effect typewriter communication with user processes should be ready by mid-February.

III. GRAPH THEORY

A. Introduction

The problems that arise in multilevel layout of electrical circuits are related to the graph-theoretical problem of extracting planar subgraphs from nonplanar graphs. While the circuit problems are both more complicated and more flexible than the graph-theoretical one, advances in applicable graph theory can produce techniques of practical value to the automation of circuit layout.

The present research has led to the development of an improved planarity test, based on a new approach to planar graphs. It should be noted that the term planarity test is used here to refer collectively to the testing of graphs for planarity, extraction of planar subgraphs from nonplanar graphs, and layout of planar graphs.

B. Description of Planarity Test

The early stages of research have already been reported.* The motivating idea of this work has been the observation that any planar graph, once a planar layout is known, can be drawn "from the interior out;" that is, starting with a single node, one can construct the graph by drawing each successive, "new" node outside the periphery of the portion completed at the preceding step; at each step, edges are added only between the new node and nodes on the periphery just mentioned. This method of drawing will subsequently be referred to as "peripheral expansion." It should be noted that this method is based on manipulation of the nodes of the graph in an ordered sequence; clearly, the ordering of the sequence is critical.

The initial research was devoted to finding a graph-theoretical property, subsequently called the connected sequence property, such that: given a graph G and an ordering of the vertices of G into a connected sequence, G is planar if and only if a planar drawing of G can be constructed by "peripheral expansion," based on this sequence. The discovery of the connected sequence property was complicated by the requirement that the peripheral attachment algorithm function without back tracking.

The most difficult portion of the research was the subsequent development of an algorithm for "sorting" vertices of a graph into a "connected sequence." The main features of this algorithm are as follows. First, a sequence of paths, called a chain sequence, is constructed. (Its definition is not important here.) The chain sequence is constructed iteratively, starting with two

* Graphics, Semiannual Technical Summary to the Advanced Research Projects Agency, Lincoln Laboratory, M.I.T. (30 November 1969), pp. 12-15, DDC AD-700316.

paths P_0, P_1 such that P_0P_1 (i.e., P_0 followed by P_1) is a circuit. In general, a partially completed chain sequence $P_0, P_1, \ldots, P_k$ is augmented in one of the following two ways:

Replace a path $P = P_i$ (where P is of the form $P^1P^2P^3$) with two paths P^1QP^3, P^2, where Q is a new path;

Insert a new path Q into the sequence, following a path P_i.

Constructing the path Q is the substance of the algorithm. The main feature of this construction is its inherently "local" character. Briefly, if we designate nodes not yet belonging to any P_j as "untreated," and paths P_k adjacent to untreated nodes as "incomplete," then path Q is generated by considering only the largest-indexed incomplete path P_i; specifically Q is constructed from the untreated nodes to which P_i is adjacent. Thus the algorithm is independent of "global" properties, i.e., properties related to arbitrary untreated nodes. Completion of the algorithm occurs when every node in the graph is included in at least one path.

Finally, a connected sequence of nodes is easily constructed from a chain sequence $P_0, P_1, \ldots P_m$ as follows. If P consists of the nodes $v_1, v_2, \ldots, v_r$, define int (P) as the path $v_2, v_3, \ldots, v_{r-1}$. Then the sequence

$$P_0 \circ \text{int}(P_1) \circ \text{int}(P_2) \circ \ldots \circ \text{int}(P_m)$$

can be proven to be a connected sequence. (If S and T are sequences of nodes, then S ∘ T denotes "the nodes of S followed by the nodes of T.")

In summary, the planarity test consists of two parts: the first is the "sorting" algorithm, which generates a connected sequence of nodes; the second is the "peripheral expansion" algorithm, which tests for planarity.

Advantages:- The primary advantages of the new planarity test are:

(1) High speed combined with a small data base
(2) Efficiency in extracting several planar subgraphs from a single nonplanar graph
(3) Simplification of the final process of coordinatizing the edges of a planar graph.

Each of these advantages will be explained briefly.

Speed and Data Base:- The peripheral expansion algorithm is intrinsically very simple and fast; at each step it operates merely on the periphery of a partially constructed graph. The speed and data economy of the sorting algorithm stem from the algorithm's depending only on local properties of the graph, as explained above.

The sorting algorithm, the slower of the two parts, has an absolute upper bound of order m^2 (m = number of edges). This bound is pessimistic, however, and in actual practice the algorithm may be considerably faster.

Planarity Extraction:- In the layout of nonplanar networks, one often wishes to extract several planar subgraphs in an attempt to best match electrical requirements. The present algorithm permits such multiple extraction with minimum effort: one can simply re-apply the highly efficient peripheral expansion algorithm, with varying heuristics for discarding nonplanar edges.

Coordinatization:– Even after obtaining a planar graph, one still must obtain geometric coordinates for the lines (and nodes) of the graph. This process is often based on a "checkerboard" or "ripple" version of Lee's routing algorithm;* which is slow and very costly in terms of storage.

The present algorithm facilitates a much simpler approach. Because the graph is constructed "from the interior out," a planar coordinatization can be obtained by using the idea of an "envelope" – an expanding rectangle enclosing the periphery of the partially drawn graph. The four points of this rectangle contain all the information needed to coordinatize the arcs connecting the current periphery with the next point in the connected sequence.

This coordinatization algorithm is suitable for rapidly generating a planar display. However, further computation would probably be necessary to increase the density and reduce the size of the graph.

Applications:– When programming is completed, the algorithm will be tested in close interaction with the needs of the Laboratory's current integrated circuits design program.

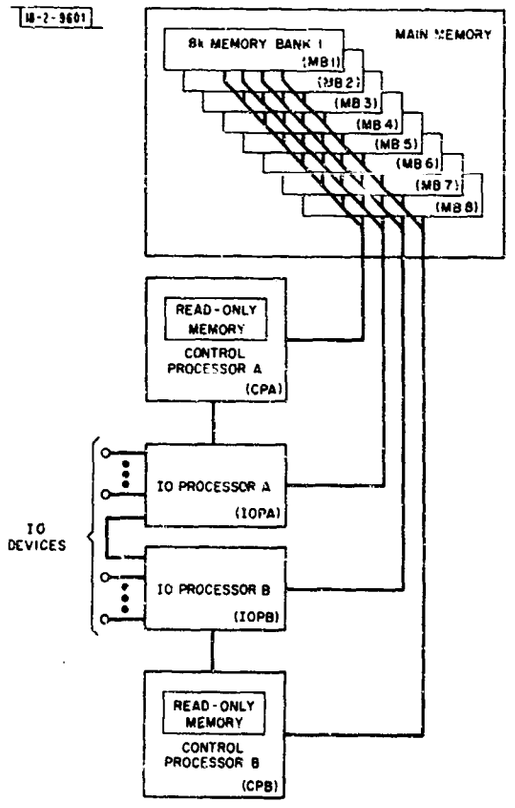

Fig. 1. TSP system hardware – dual Meta 4 processor configuration.

* C. Y. Lee, "An Algorithm for Path Connections and Its Applications," IRE Trans. Electronic Computers EC-10, 346-365 (1961).

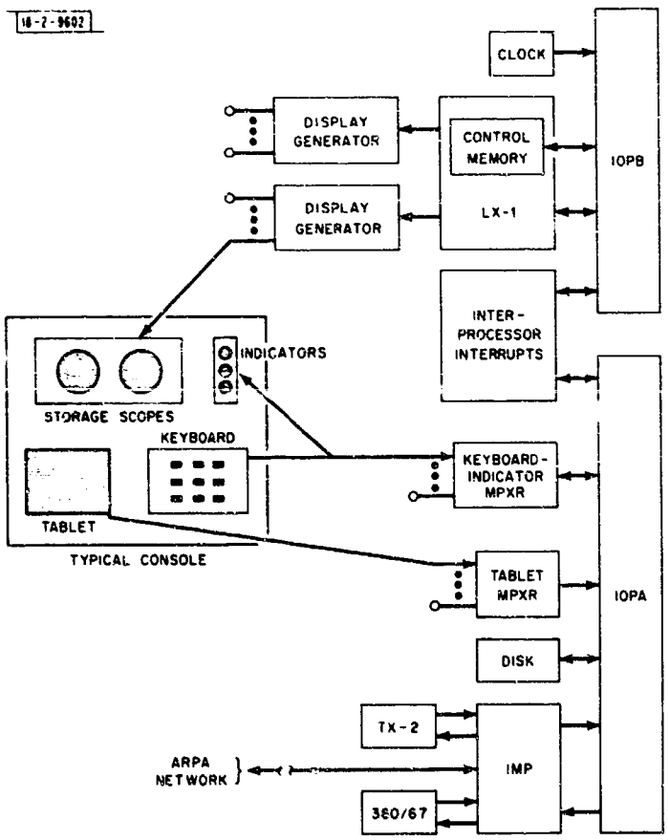

Fig. 2. Principal IO devices in TSP system.

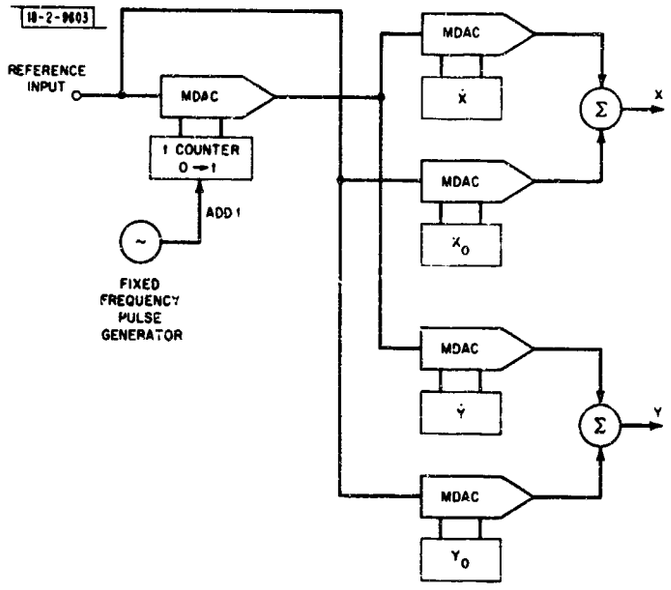

Fig. 3. TX-2 display generator (vector generator portion).

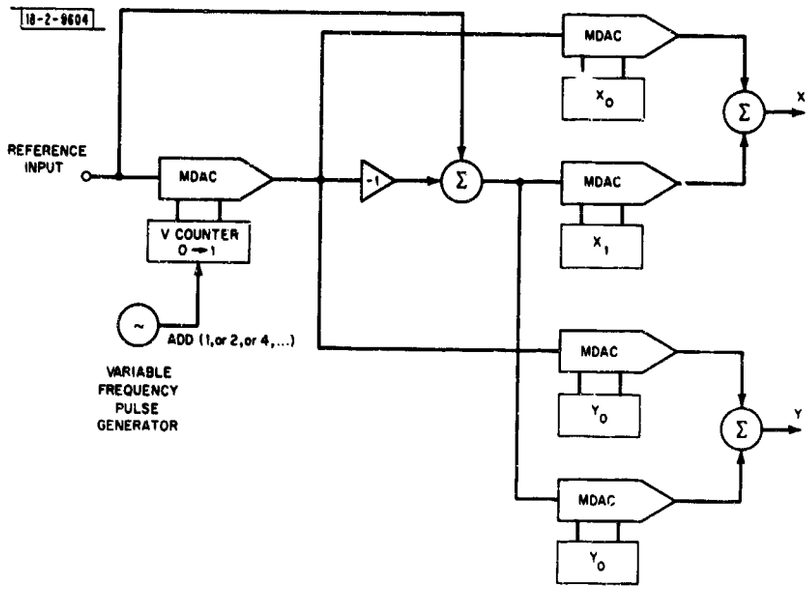

Fig. 4. TSP display generator.

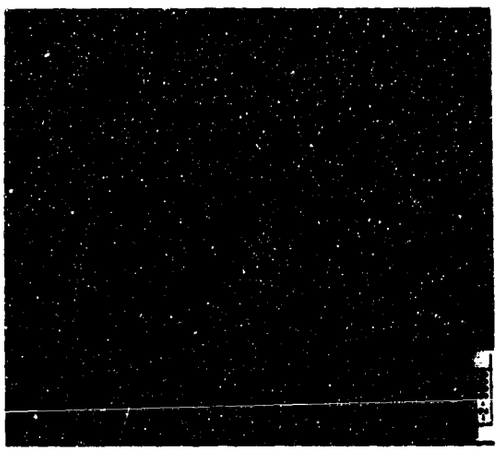

Fig. 5. TSP character set, normal mode, photographed from Tektronix 611 storage scope driven from TSP character generator connected to SEL 810A test computer.

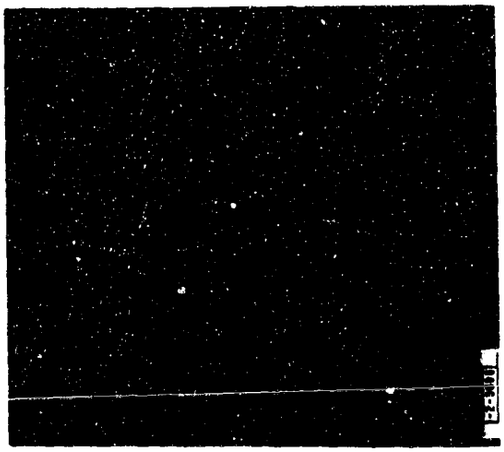

Fig. 6. TSP character set, slant mode, same conditions as Fig. 5.

Fig. 7. Sample of text by TSP character generator, same conditions as Fig. 5.

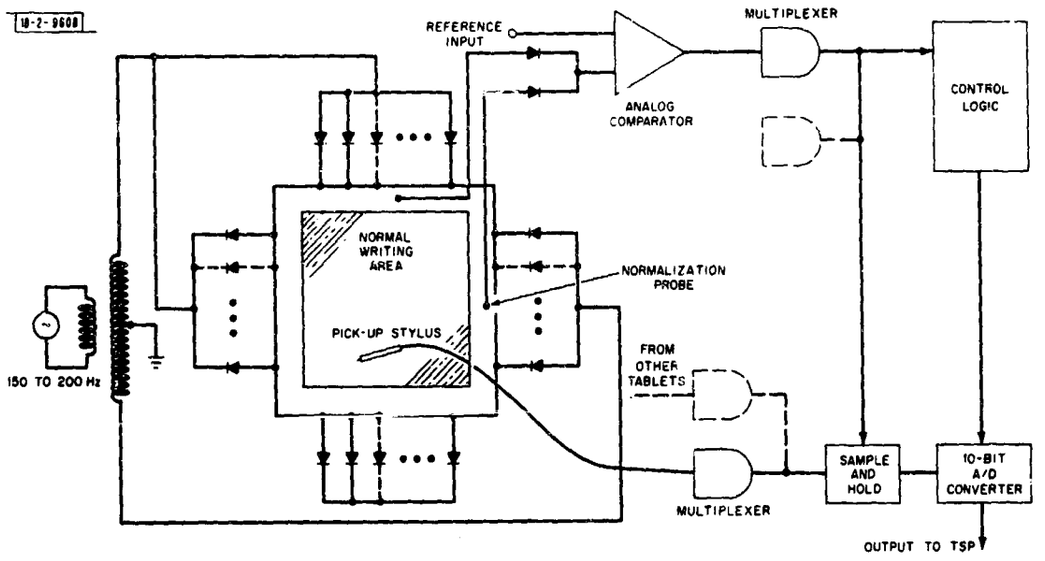

Fig. 8. TSP tablet system.

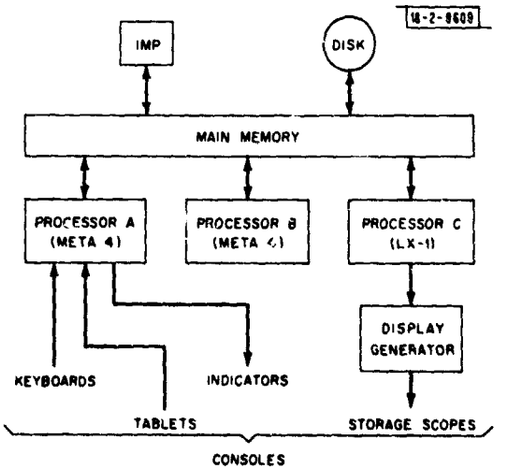

Fig. 9. Simplified diagram of TSP system.

DOCUMENT CONTROL DATA – R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Lincoln Laboratory, M.I.T.	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP None

3. REPORT TITLE

Graphics, Semiannual Technical Summary to the Advanced Research Projects Agency

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Semiannual Technical Summary – 1 June through 30 November 1970

5. AUTHOR(S) *(Last name, first name, initial)*

Forgie, James W.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
30 November 1970	28	3

8a. CONTRACT OR GRANT NO. F19628-70-C-0230 b. PROJECT NO. ARPA Order 691 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) Semiannual Technical Summary for 30 November 1970 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* ESD-TR-70-355

10. AVAILABILITY/LIMITATION NOTICES

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	Advanced Research Projects Agency, Department of Defense

13. ABSTRACT

System architecture for the Terminal Support Processor (TSP) system is now complete. The principal element in the system is a dual processor version of the Digital Scientific Corporation Meta 4 microprogrammed computer. Main memory (magnetic core) consists of 8 banks of 8K 16-bit words. Secondary memory is provided by a one-million-word Digital Development Corporation disk having a 495-kHz transfer rate and an average access time of 8.7 msec. The TSP is connected via an Interface Message Processor (IMP) to the ARPA computer network and other Lincoln Laboratory computers. Consoles consist of a keyboard, a small number of visual and audible indicators, a graphic input tablet, and a pair of storage scopes. Displays are driven from a shared vector and character generator which is now complete and being tested. An extensive character set includes most of the commonly used sets. A low cost input tablet has been developed utilizing a resistive sheet as the writing surface and an ordinary pencil as a pickup stylus.

TSP system software will be written in an extension of BCPL. BCPL programs will be compiled into code for a 16-bit machine called BCOM which will be simulated by microcode in the Meta 4 processors. The BCOM machine has an unusual addressing scheme oriented toward the addressing modes most often used by the BCPL compiler. Much IO programming in the TSP will be handled directly in microcode, and the processors in the system have been assigned specific system tasks to take advantage of the speed potential of specialized microcode. A simulator and assembler for Meta 4 microcode are now available on TX-2.

An IMP for connecting TX-2, the IBM 360/67, and the TSP to the ARPA network was delivered in July 1970, but connection to the network awaits installation of telephone equipment now scheduled for December 1970. The TX-2 interface hardware is connected and operating. The 360/67 interface is scheduled for delivery in December. Network control programs for both machines are under development.

Graph-theoretical work has yielded an improved planarity test algorithm. Its advantages and utility will be tested in close interaction with the Laboratory's integrated circuit design program.

14. KEY WORDS

graphical communication
TX-2
time sharing
man-machine
display systems
programming languages