

EXPERIMENTER'S COMPUTER SYSTEM

by Carl T. Helmers, Jr.

Part 1: Design Goals and Introduction

One definition of progress in technology might be the following: "creation of new technological areas and the reduction in cost of older technologies." As a reader of this article, you are undoubtedly interested in the prospect of acquiring and using a fully programable general purpose computer system at a reasonable cost. The recent advances in microcomputer technologies fit the above definition of technological progress by making possible a general purpose bus-oriented minicomputer CPU (old technology, lower cost) packaged in a relatively small number of integrated circuit packages (new LSI technology.) This particular advance for the first time makes it possible for the computer enthusiast, educator or hobbyist to consider building and using a computer system at a reasonable cost. The purpose of this series of articles is to provide a thorough exploration of the possibilities inherent in LSI microprocessor technology as a vehicle for building a computer and incidentally teaching computer systems design principles and software design techniques.

The first installment of the Experimenter's Computer System series is intended to serve as an introduction to the project. It outlines the scope of the project, its design philosophies and what you — the computer enthusiast and potential builder of such a system — will learn by reading these articles and using them as a source of ideas for your own work. The results will of course be rewarding in proportion to the time and effort you put into the project. The modular nature of the Experimenter's Computer System allows you to take many options in customizing a system to your own personal needs. In fact, once the basic components of the system are assembled, the addition of more memory, I/O channels, peripherals and other components is effectively limited only by time and your own budget. This general modularity is a consequence of the bus oriented design of the micro-computer selected for the Experimenter's Computer System.

1. OPTIONS:

Before going into the details of the Experimenter's Computer System it is worthwhile to consider the various options which are open to the individual or organization operating on a limited hardware budget. The design found in these articles is one of the most economical ways to acquire a computer in terms of dollar amounts. However its relatively lower cost must be traded off against the time required to construct and test the various modules.

1.1 PURCHASING OR RENTING A COMPUTER SYSTEM

This option is the most expensive of the group considered here. By purchasing or renting a complete computer system, the time involved in building and debugging system hardware is avoided, but a higher dollar

price is the result. Incidental to the process of buying a complete system, little is learned about the hardware design aspects of computer systems. The author is familiar with several systems which may be rented at prices in the \$250 to \$400 per month range, or purchased for prices in the neighborhood of \$10,000. Such systems are typically complete general purpose computers with integrated keyboard, CRT display and mass storage — eminently suitable for use in teaching the principles of software design and programming while incidentally providing a useful tool in handling personal business, calculation and record keeping. The rentals and purchase prices of such systems are however outside the range of most individuals.

1.2 PURCHASING A MINICOMPUTER MAINFRAME CPU

The purchase of a minicomputer CPU outright is one way in which to bypass much of the effort required for construction of a system from scratch. The prices are however generally higher than for the micro-computer oriented system described in this series of articles. For example a typical new 16-bit minicomputer with 4096 words of storage, hardware multiply/divide and an excellent instruction set costs about \$2000 in a table top package. A used computer can also be acquired, at a lower cost (eg: about \$1200 or so depending upon the machine.) In either case, this particular method of acquiring a computer gives the purchaser a head start at a correspondingly higher cost. The engineering of inexpensive peripherals described in this series of articles is still valid, and can be adapted to the requirements of any minicomputer without great difficulty.

1.3 THE MICROCOMPUTER OPTION

The method of acquiring a computer system described in this series of articles is to employ a micro-computer LSI integrated circuit as the CPU and static MOS memory for programs and data. With this main frame for the machine, a set of inexpensive peripherals completes the system. By choosing this course, a low basic cost is achieved at a price in the time required to build a complete system. The cost of the standard Intel MCS-8 computer CPU part used in this design is quoted at \$120 by one nationwide electronics distributor's catalog; the judicious selection of components for memory and peripherals can produce a working system with a complete cost in the range of \$500-700 using all new parts.

1.4 THE COMPUTER DESIGN OPTION

On a cost basis equivalent to or slightly less than the micro-computer option, it is possible to completely design and build a computer CPU from scratch. For those individuals with an extremely limited budget a very simple design for a CPU with correspondingly limited capabilities can be built using perhaps \$50 to \$100 worth of parts total. However, such a path is probably most appropriate for an engineering course in computer design where the goal must be limited in scope. It is frequent practice to take this option in many engineering schools. This option suffers from the disadvantage of simplicity. To implement a reasonable instruction set for such a computer would involve sufficient complexity to make the microcomputer option more attractive due to cost. As a result, a simple processor designed and built completely from scratch does not have the generality and expansion capabilities of a microprocessor.

2. SOME DESIRABLE CHARACTERISTICS:

2.1 STANDARD PARTS

The Experimenter's Computer System should be oriented towards the utilization of standard electronics parts. By using commonly available parts the problems of a one-of-a-kind approach — scarcity and uniqueness — are minimized. Such parts are in general the least expensive due to competitive forces and mass production. Due to widespread usage the integrity of the parts design can be assumed in general. This consideration of standardization extends to the choice of a CPU in this era of LSI — the INTEL MCS-8 chip which forms the CPU of this design is a standard distributor catalog item which is well proven in wide application since its introduction in late 1971.

2.2 SIMPLICITY

The concept of simplicity comes in two packages when it comes to the design of computer systems. First there is the simplicity of the basic design in terms of the number of physical hardware components which must be assembled and checked out. Second, there is the simplicity of the programming which can be helped or hindered by the design of the computer's instruction set — its "architecture." Both kinds of simplicity are desirable in order to obtain a practical computer system.

2.3 MODULARITY

The design of a computer system should similarly reflect the principles of modularity in two areas: first, the hardware should be modular so that it may be built and debugged in stages, with the interaction of modules limited to carefully defined interfaces. Secondly, the software of the system should be modular so that programs and systems of programs may be built and debugged in a similar manner. Both aspects of modularity are widely used in the computer industry as a result of the greater efficiency of design, debugging and configuration possible through this approach. The practical use of this principle is assumed throughout the articles of this series, at early stages in the hardware design, and in later articles concerning software design and "structured" programming.

2.4 FULL PROGRAMMABILITY

The design of the Experimenter's Computer System should be a fully programmable general purpose computer with read-write memories throughout. One of the main purposes of designing a fully programmable computer (as opposed to a series of special purpose devices with limited generality) is to take advantage of this programmability. If a new function is desired from the device, merely writing a new program may be sufficient to achieve the desired performance from the system. For instance, to change a calculator program's display from fixed decimal (the easiest to implement) to full floating decimal or even scientific notation is accomplished by the modification of the software involved, not by the substitution of new hardware components.

The use of read-only-memory (ROM) programs is thus confined to one specifically limited area: the cold start program needed to initialize the computer and branch to a particular memory address where executive and service routines are located. While ROM modules could be substituted for general read-write memory anywhere within the computer, it is assumed in this design that full programmability should be retained by excluding Read-Only-Memories.

2.5 PERIPHERALS

A computer by itself is a useless piece of machinery — it must have a set of peripheral devices with which to communicate to the "outside world." Peripherals bear the same relationship to the computer processor that sense organs and muscles bear to the human brain. In the course of this series of articles on microcomputer systems for computer enthusiasts, a fairly large emphasis is placed upon the design and construction of inexpensive peripheral hardware:

2.5.1 Mass Storage: The design must include some provision for a mass storage device so that data can be retained permanently off line for purposes of keeping programs and data available for future use. This is one peripheral which can not be omitted. The ideal form of mass storage in the context of the Experimenter's Computer System must be inexpensive, reliable and easy to construct. As a technology which meets these requirements, the audio tape recording medium is chosen for this purpose: a battery driven audio cassette tape recorder can be used to store digital data at a rate of 100 baud (1 baud = 1 bit/second.) High quality reel-to-reel and cassette recorders can store the data at a much higher rate (eg: 300+ baud.) In either case, a simple FSK (frequency shift keying) method of modulation is used. Output signals are created by selecting one of two digitally generated frequencies; input signals are detected by a phase-locked loop with an asynchronous data-generated clock. The basic I/O capability for mass storage for a minimum configuration of the Experimenter's Computer System is a single channel; additional channels may be added in parallel by adding more modems, or on a shared basis through logic required to select one of several tape recorders.

2.5.2 Displays: The subject of inexpensive displays leads to a variety of possibilities. Since the system design is modular, any and/or all of the suggestions shown below are potentially useful:

a. Binary Lamp Display: This form is the most limited in function and convenience, but is the simplest to implement. It will be one of the first display's to be incorporated in the Experimenter's Computer System.

b. Decimal Numeric Display: This type of display is the least expensive form which offers more utility than the simple binary lamp display. Its level of complexity is not much different from the binary form — consisting of the addition of BCD-Seven Segment decoders to the parallel binary words required for the simpler binary display. Software can be used to achieve BCD, octal, hexa-

decimal , floating point BCD or scientific notation BCD outputs in such a display.

c. Text Display: There are several possibilities for the implementation of a textual display. These include building the refresh memory and ROM dot-matrix character generator needed to drive a television, and purchasing Burroughs plasma displays (32 characters for \$168). Any display device which will accept serial character outputs of 6 to 8 bits in width can be potentially interfaced with this system.

d. Oscilloscope Graphics Display: A graphics display is used for drawing "pictures" under computer control. For those individuals with an X-Y oscilloscope one of the simplest forms of graphic display possible is one involving an oscilloscope and three digital-to-analog output conversions driven by the computer. One DAC channel is used for each of the X and Y inputs of the scope; the third DAC output is used for intensity modulation. This form of display is essential for the programming of a "space war" game, the "game of life" or other interesting recreational activities which use the computer system as a central theme.

Other display options are as numerous as the number of display technologies available. All are potential outputs of the Experimenter's Computer System.

2.5.3 Keyboards: The inverse of a display operation is to accept input from the human operator of the system using keyboards or other arrays of switches. As is the case with displays, there are many options available ranging from a simple switch array for entering and editing programs to ASCII typewriter keyboards needed for more advanced software techniques and textual data entry. Custom keyboards can be built and integrated on short notice for special purposes. For instance a "Space War" program will require a special control panel built for each player with a common oscilloscope screen as the "solar system" map display. Similarly, a calculator program interfacing with a BCD numeric display or text display will in general require a numeric keyboard for entry and a function keyboard for selecting the operations desired. Using the Experimenter's Computer System the programming of a variety of special purpose calculator functions of a statistical and mathematical nature can be achieved.

2.5.4 Special Purpose Peripherals: In addition to more or less standard peripheral devices the Experimenter's Computer System design supports the concept of special purpose I/O devices. For instance it is possible to interface a series of Digital-Analog converters to the processor in order to control the timing of voltages in a scientific experiment. Such DAC outputs can be used to control any fundamentally analog process requiring voltage-level settings. Similarly, the inverse of DAC outputs is an input ADC conversion. Fairly simple and inexpensive ADC channels can be constructed using tracking converters to read voltage levels as binary numbers. (See M.P. Publication #73-2 for ideas on DAC and ADC conversions.) Similarly, discrete single-bit digital inputs and outputs can be created through simple parallel interfaces to the bus.

2.6 SOFTWARE AVAILABILITY

The use of a standard computer part for the Experimenter's Computer System described in this series of articles is desirable from several points of view:

a. The manufacturer of the computer part has a wide following of customers developing programs using the same basic instruction set. Many of these programs contain interesting and useful techniques which may be incorporated into customized programs of the individuals building the system. To the extent that the manufacturer of the computer part makes such routines contributed by its customers available on a general basis, the programming problems for individual experimenters are simplified.

b. Applications program developed explicitly for this computer project by the author and his associates can be used directly by any individual building similar equipment. Descriptions of such programs provide the theme for many of the articles in this series following an initial concentration on the minimum hardware requirements in the first few articles.

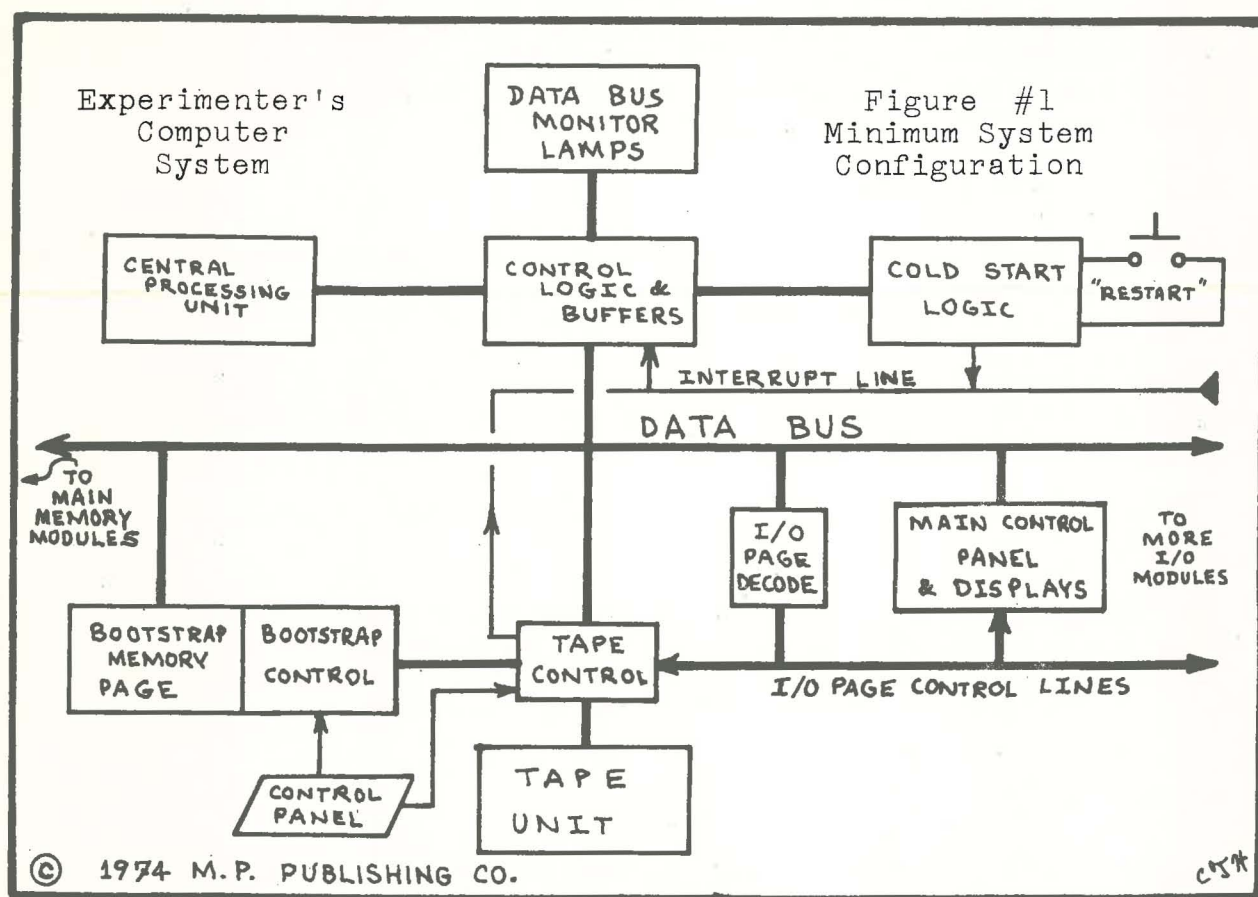
2.7 ACHIEVABILITY

As a final consideration for the Experimenter's Computer System, the criterion of achievability is essential. The implementation of a digital computer is not a trivial undertaking — especially when attempted by the individual experimenter or small group of computer enthusiasts. A prime purpose of this design and the series of articles based upon it is to come up with a computer system of useful capacity which is achievable at moderate cost by any individual seriously interested in the subject. The use of a standard micro-computer part is a state-of-the-art short cut which simplifies the project immensely. Even so, full implementation of the design in these articles will require care, diligence and persistence on the part of the builder. With the articles in this series as a guide, the individual experimenter or small group of computer enthusiasts now have a means to create a general purpose computer system and in the process learn a great deal about computer hardware, software design principles and computer technology in general.

3. HARDWARE SYSTEM DESIGN:

The microcomputer chip chosen as the basis for the Experimenter's Computer System is a bus oriented central processor designed and manufactured by the Intel corporation: the 8008 processor first announced in 1971. This microcomputer chip has attained widespread acceptance in the digital systems industry due to the fact that it is TTL logic compatible, has an instruction set comparable to many minicomputers, has a general purpose bus-oriented data architecture which is inherently modular, includes special purpose program control mechanisms designed for modular programming and — most important — is available at a relatively low cost. All of these advantages make it a desirable machine for use by computer enthusiasts interested in acquiring an inexpensive system.

The basic block diagram for the Experimenter's Computer System is shown below in figure #1. Central to the design is the 8-bit parallel data bus and its related control lines. This bus is bi-directional so that transfers both to and from the CPU are accomplished on the same set of 8 lines. Due to the bus orientation of the system there is an inherent hardware modularity, symbolized by the arrows extending the bus and control lines off the page. This makes the system capable of growth and expansion on an incremental basis: the establishment of a minimum hardware configuration serves as the basis for further expansion and improvement as more I/O and memory modules are added. Figure #1 represents a minimal system which serves as the first milestone in the construction of an Experimenter's Computer System:



3.1 CENTRAL PROCESSING UNIT

The CPU is the microcomputer system's most fundamental component, the Intel 8008 processor chip. This device is a complete general purpose minicomputer type machine with an 8-bit parallel data architecture, 7 internal registers, a 7-level program control stack, 45 instructions and addressability of 16,384 bytes of memory. The clock rate of the standard Intel part is 500 KHz, which means that the typical instruction takes 20 microseconds depending upon the number of clock cycles required for completion. This speed is not particularly fast — it is approximately the same speed of execution found in the onboard computers of the Apollo spacecraft which were used to navigate to the moon and back.

However, a 20 micro-second instruction time is more than adequate speed for most uses of the Experimenter's Computer System, where — as in the process of navigating a space ship — answers are more important than the time it takes to compute them (within limits of course!)

3.2 CONTROL LOGIC AND BUFFERS

This block is incorporated in the design for several purposes. First, it is desirable to isolate the CPU chip itself so that it is protected from stresses beyond its ratings. Second, the CPU requires an external clock at 500 Khz in order to operate, provided by an appropriate set of clock generation logic elements. Third, the control of the entire computing system must be decoded.

The data bus shown emanating from this block is a tristate TTL bus employing Signetics 8T09 interface gates. A maximum of 24 modules may be hung on the bus without resorting to additional buffering. The bus time multiplexes data and address traffic in 8-bit segments under the control of the CPU's timing signals.

3.3 DATA BUS MONITOR LAMPS

This logic is provided so that the data bus can be monitored for debugging and educational purposes. The logic included for this purpose will enable the following forms of monitoring:

- a. Selective monitoring of data at specific processor state times in the multiplex scheme, via decoding of the processor state information.
- b. Real time monitoring of the bus independent of the processor state.

The panel readouts for this function consist of a set of 8 LED indicator lamps and form the first display unit to be included in the system under computer program control. Panel switches will include a rotary state selector switch for the first mode, and a two-position switch to select modes.

3.4 MEMORY ADDRESS SPACE ALLOCATION

The memory address space of a digital computer is the set of binary integer numbers which are valid addresses for memory operations. The memory address space of the Intel MCS-8 system is thus the set of numbers 0 to 16,383 which are the potential contents of its 14-bit program counter and stack mechanisms. This space may be conceptually divided into 64 "pages" of 256 bytes per page by the time-multiplexed division of a 14-bit address into an 8-bit low order component and a 6-bit high order component. The 6-bit high order component is the "page address" and the 8-bit low order component can be termed the "byte address." The nature of the memory devices connected at each page address of this design can be fairly arbitrary since the CPU cannot distinguish between a read-write register in an I/O device and the ports of a read-write random access memory. The CPU is completely independent of the particular timing constraints of memory devices as well, since it can wait indefinitely for the "memory ready" signal to be indicated. For the purposes of the Experimenter's Computer System, the memory address space is given the following fixed allocations:

3.4.1 Bootstrap Page: In order to establish a basic kernel of systems software in a system which depends on such software for its operation, a means is required to "bootstrap" an initial program load (IPL). The means of doing this in the Experimenter's Computer System is a special page of memory at addresses 3F00 to 3FFF (hexadecimal notation.) This page is characterized by two manually selected modes of operation:

- a. "Normal" computer-controlled operation in which this page acts like any other main memory page of the system, containing either program or data storage.
- b. "Bootstrap" manual operations in which the memory is controlled by hardware logic of the "Bootstrap Control" module. In this mode, a special control panel and the tape recorder I/O unit are connected to the bootstrap memory. Operations include manual "toggling in" of programs and data, examining and changing data manually, dumping the IPL program to tape, and restoring that program from tape.

Since the bootstrap page is subject to control by a separate manual control panel independent of the CPU, it is possible to manually load and alter its 256 bytes of memory. This page is intended to be used for systems software: routines used as tools for the development and debugging of further programs. The following list represents a minimum set of such bootstrap systems routines:

- a. Memory Dump/Restore Routines.
- b. Display, Debug and Edit Routines.
- c. Tape Recorder I/O Control & File Management.

Depending upon the particular memory space requirements of these routines, additional programs for functions such as extended precision arithmetic, and block data movement might be incorporated in this page.

3.4.2 I/O Page: In order to simplify the hardware and programming of the Experimenter's Computer System, input/output operations can be treated conceptually as a special kind of memory connected to the bus. One page of the memory address space is reserved for use in I/O operations. This page is allocated addresses 3E00 to 3EFF (hexadecimal notation) and is thus the next-to-highest page address in the system. Not all of the 256 addresses will necessarily be given an actual hardware implication — unused addresses with no attached device result in a null data configuration (all "1" bits on the bus) and are treated as "HALT" instructions if the processor should by chance jump to such a location. In the minimum configuration of figure #1, only the addresses connected with the main control panel, control panel displays, and the tape recorder programmed I/O channel are active. As modules are added to the system, new addresses on this page may be utilized for the new functions: sharing the common I/O Page decoding logic, attaching directly to the bus for data transfers, and employing local logic to decode the byte address.

3.5 MAIN CONTROL PANEL/DISPLAYS

The minimum system configuration for the Experimenter's Computer System shown in figure #1 includes a control panel and displays driven by the systems software which is loaded into the bootstrap memory. This panel consists of the following:

- a. 14-bit binary LED display for addresses. This output is allocated 2 byte addresses at hexadecimal 3E00 and 3E01 in the I/O page.
- b. 8-bit binary LED display for data. This output is allocated one byte address at hexadecimal 3E02 in the I/O page.
- c. Hexadecimal data entry keyboard array (16 pushbutton switches) and an auxiliary array of 16 function switches. Control logic is used to encode two 4-bit patterns in a single word allocated at one byte address of 3E03 in the I/O page. Pressing a key writes data into the appropriate memory word; program acknowledgement and resetting is accomplished by storing a null pattern to reset the word. The first four bits of the word (high order) are reserved for the function switch selection; the second four bits are used for the data switch selection coding.

3.6 TAPE MASS STORAGE

The mass storage function of the Experimenter's Computer System is performed by using a magnetic tape recording medium. In figure #1 this function is represented by two blocks:

- a. The Tape Unit is a serial modem and control logic used to write and read serial blocks on an Audio Cassette Mass Store.
- b. The Tape Control block of the diagram is designed to provide dual control functions for the Tape Unit: A fixed length IPL dump/restore mechanism is used in the Bootstrap mode; a programmed I/O mechanism run by the CPU and its interrupt mechanism is used after IPL operations are completed and the bootstrap memory page is properly set up with basic systems routines.

3.7 COLD START LOGIC

The central processor requires a special set of instructions to be executed in order to start up the system for the first time or to re-initialize the system at a later time. This is provided by logic which disables normal bus activity and forces a fixed program load of several instructions to be executed following the interrupt generated by a restart switch. The major function of this program is to jump to the beginning of the bootstrap memory page (address 3F00). A 32x8 ROM is used for this program — an 8223 IC or a reprogrammable simulation of this circuit. In the later stages of this project, the cold-start program will be improved to include automatic sequencing of bootstrap memory restoration prior to jumping to the beginning of bootstrap memory.

This concludes part 1 of the Experimenter's Computer System project. The next article, part 2, describes the Tape Unit.

AUDIO CASSETTE MASS STORAGE SYSTEM

by Carl T. Helmers, Jr.

This publication is the second in a series devoted to the proposition that computer technology is within the price range of the serious experimenter and computer enthusiast. In the first instalment of this series a discussion of the goals, design tradeoffs and the overall system design were presented. In the current article, the first construction project in the series is described: the Audio Cassette Mass Storage System which will be used to permanently record and store both programs and data.

This article describes an audio tape cassette interface which is capable of reading and writing data at 100 bits per second on an inexpensive recording medium. The use of a device of this type is a necessity in the Experimenter's Computer System if it is to be conveniently used and programmed. If two or three of these I/O devices are constructed for the system, some fairly powerful file-handling applications for the computer will be programmable.

The information presented here describes the basic serial I/O unit as designed and built by the author. The unit accepts serial TTL data from a controller in order to write frequency shift keyed (FSK) data on the tape device. For input of the same data, the unit demodulates the audio signal with a Phase Lock Loop to provide a serial data stream to the controller.

The primary goal of this particular design effort was to duplicate the functions of a paper tape reader/punch of the type found in the typical minicomputer installation. As a bare minimum these functional characteristics are:

- 100 Baud (bit/sec) data rate
- Permanent storage
- Reasonable reliability
- Manual control of the I/O functions with the exception of start and stop.

This design accomplishes these goals to provide the first peripheral device for the Experimenter's Computer System. Some comments on improving the performance of the device are included at the end of this article, however the design as it stands is functional and proven in operation.

1. THEORY OF OPERATION:

1.1 SYSTEM CONTEXT:

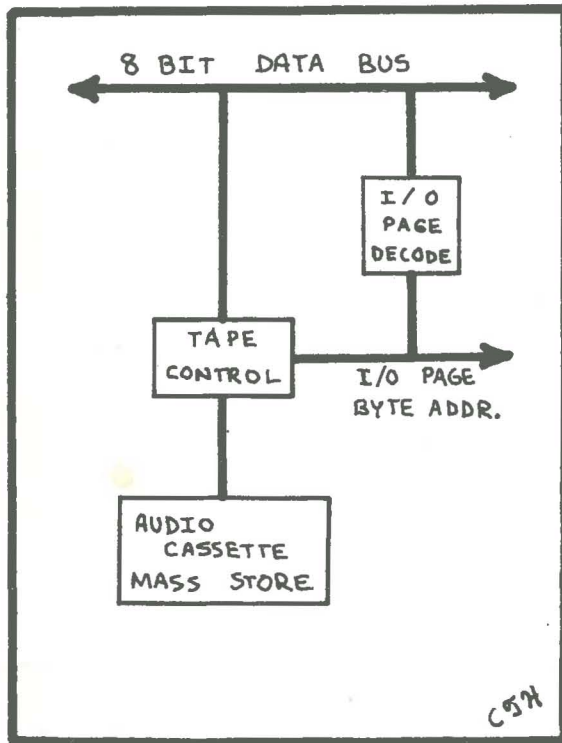


Figure 2. System Context

The Experimenter's Computer System defines the context in which this I/O device is used. The full description of the Experimenter's Computer system was outlined in Part 1. Figure #2 at the left is an adaptation of part of the System Block Diagram found in ECS-1 on page 7, concentrating on details of the tape interface.

The I/O Page Decode logic is shared by all the I/O devices which are accessed by memory operations as opposed to I/O instructions. Its purpose is to decode the high order page address provided by the CPU and to store the low order byte address for decoding by various devices. The Tape Control unit contains the logic for interfacing the Audio Cassette Mass Storage System to the CPU data bus. The major part of this interface is the 8-bit parallel to serial data format conversion required.

Details of the Tape Control, I/O Page Decode and other components of the computer main frame are the subject of later articles in this series. To understand the operation of the Audio Cassette Mass Storage System device requires no further consideration of the particular computer and controller which will utilize it.

1.2 TAPE DATA CONCEPTS:

The basic method of storing data chosen for this system's design is to frequency modulate an audio signal by means of a digital switch. In order to keep frequency deviations due to "wow and flutter" of the tape mechanism small compared to the actual encoded signal, and additionally to provide an easily generated ratio, an octave (2:1) frequency separation was chosen for the information. For the 100 baud data rate used in this design, the lower frequency, "f", is 3000 cps and the higher frequency, "2f", is 6000 cps. There is a 1:1 correspondence between the frequency on the tape recording and the TTL logic level of a data bit internally: f is understood to be the logic 0 signal, and 2f is understood to be the logic 1 signal. All clock and timing involved in this system (with the exception of two start up delays) are defined in terms of f, as can be seen in the notations at various points on the detailed logic diagrams of figures #6a and 6b.

1.2.1 DATA FORMAT:

Figure #3 found on the next page gives the details of the self-clocking data format used in this design. The drawing is a timing diagram

for several of the signals in the system. Time is interpreted as the common horizontal axis, with digital signal levels represented by vertical displacements of the appropriate lines. The diagram of figure #3 covers two bit periods (t_{bp}) as recorded on the tape or read from the tape.

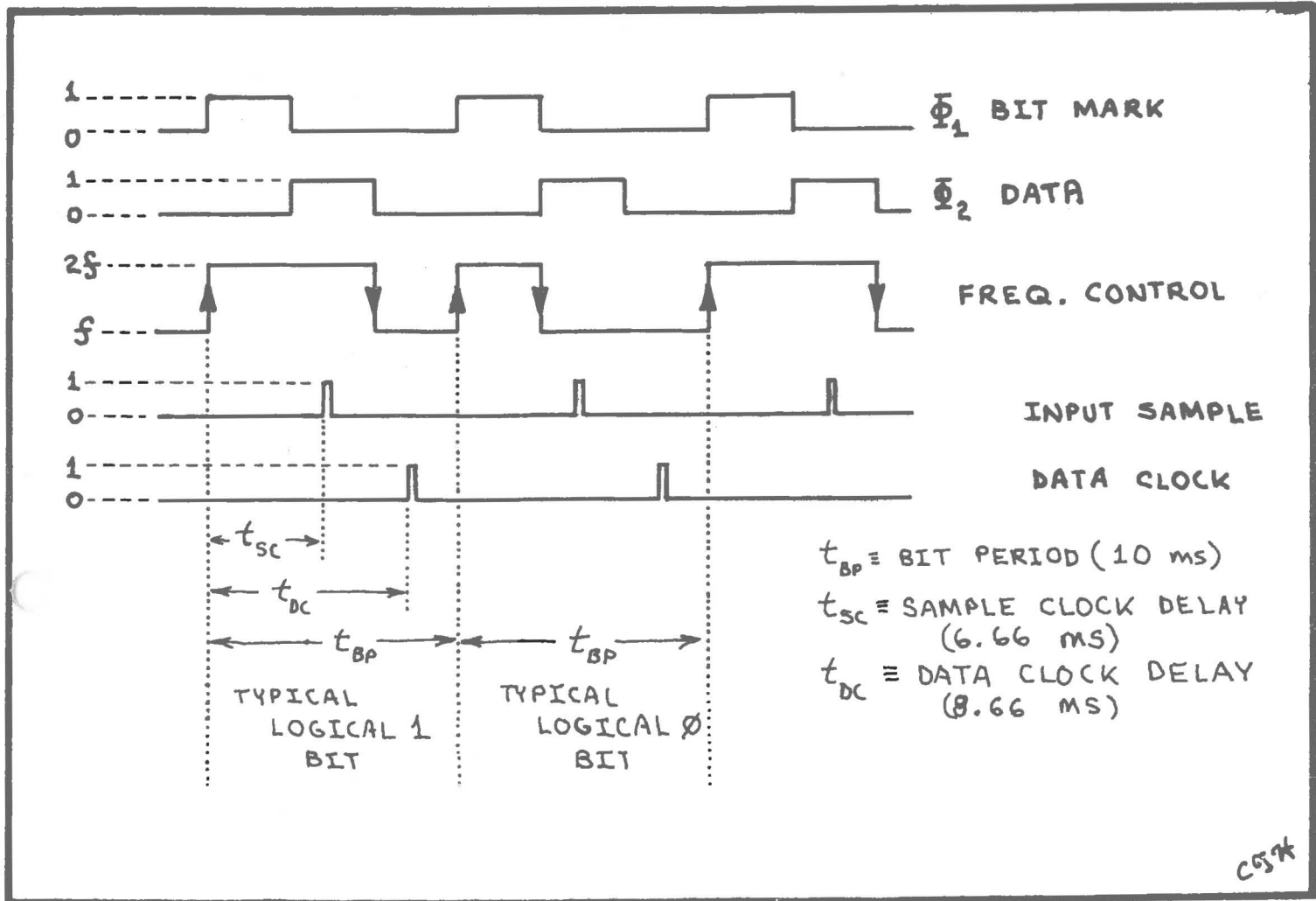


Figure 3. Tape Data Format

The concept of a self-clocking data format is this: each bit period of information recorded includes data plus information required to regenerate timing of the period. In this way the process of reading the data can be made independent of timing variations in an imperfect recording medium. In the technology of information transmission, the most widespread example of a self-clocking data format is the modulation used for television video information: the synch information is sent along with the analog data signal and the receiver locks onto this clock.

The self clocking nature of the data format used in this design is found in the number of state transitions per bit period. A bit period is the basic unit of time required to record a single bit of information on the tape. For a 100 baud rate there are 100 bit periods per second with each bit period taking 10 milliseconds. The format chosen here

uses two transitions in frequency per bit period, as shown by the arrows in the "Freq. Control" signal shown in figure #3. There is a transition from f to $2f$ followed by a second transition from $2f$ to f for each bit. By picking the f -to- $2f$ transition as the fixed reference point in the period it is possible to regenerate input timing information synchronized to the actual data signal.

The bit period (t_{BP}) of 10 milliseconds is divided into three parts to format each bit of information. These parts are:

- Bit Mark phase: The first part of the period (Φ_1) is the Bit Mark phase, represented on the tape recording by the frequency $2f$. The beginning of this phase is the f -to- $2f$ transition which marks the start of a bit.
- Data phase: The second part of the period (Φ_2) is the Data phase. Data is represented on the tape during this portion of the period by the choice of f (logical 0) or $2f$ (logical 1). In terms of transitions of frequency, if the $2f$ -to- f transition occurs at the end of the Data phase then the data is a logical 1; if the $2f$ -to- f transition occurs at the beginning of the Data phase then the data is a logical 0 for the bit in question. For purposes of inputting data, sampling the state of the demodulated signal during the the Data phase (eg: at t_{SC}) is a timing requirement of the data format.
- Null phase: In order to provide for the f -to- $2f$ transition which marks the start of the next bit period, there must be a Null phase at the end of each bit period, during which the frequency f is always recorded.

The width of all three phases is set identically in this design by dividing the bit period into 15 equal parts with a counter and some phase generation logic.

In addition to the Input Sampling clock (see figure #3), there is a second clock referenced from the beginning of the bit period. This is the Data Clock which is used during both input and output to indicate a request for more data (output) or to cause storage of the data just read (input.)

1.2.2 BLOCK FORMATS:

Figure #4 (next page) shows the Tape Block Format used to group a series of recorded bits of the type just described. The concept of a "Data Block" is defined as a contiguous series of bits written or read as a logical unit in a single operation. The concept includes the information bits plus any "overhead" information required to physically define the data. In this system, the overhead consists of the wasted tape (inter-record gap) between the end of one block (while the recorder slows down) and before the first bit of the next block (after the motor has gotten up to speed and stabilized.) One such block is read or written whenever the Audio Cassette Mass Store receives its I/O Start cue from the controller. Figure #4 is also in the form of a timing diagram for several signals with a common horizontal time axis:

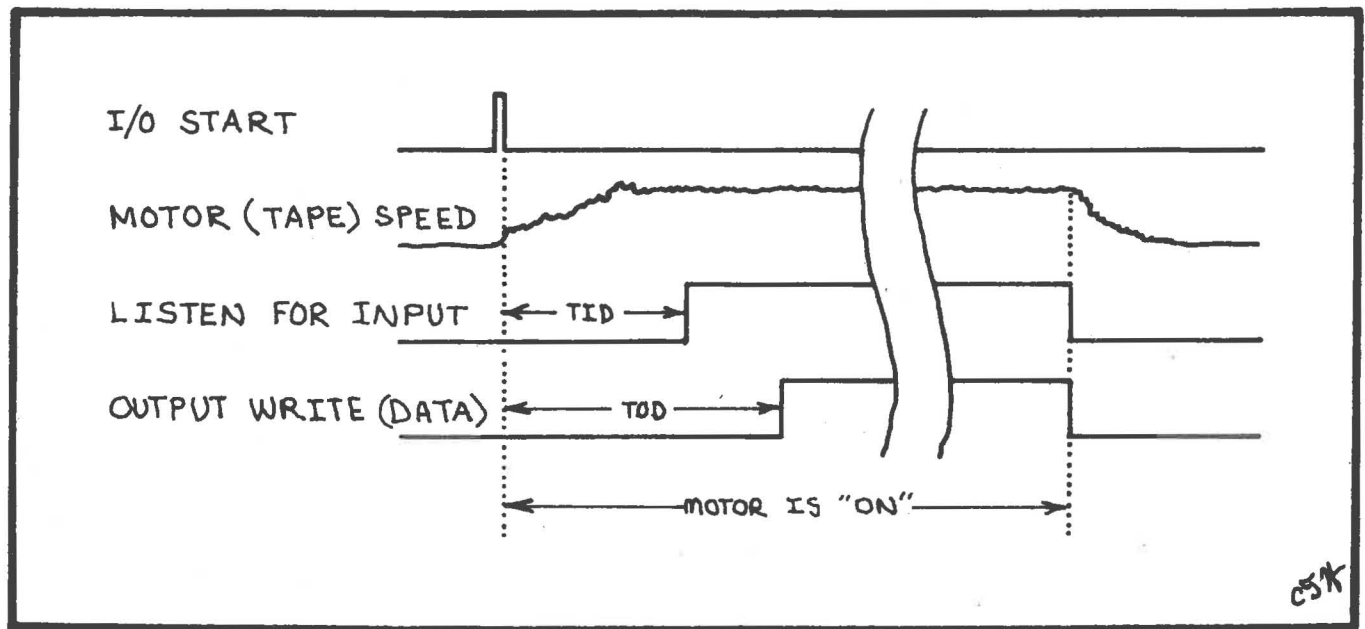


Figure 4. Tape Block Format

The "I/O Start" pulse shown at the top of the diagram is a signal defined by the Tape Controller and used to initiate an I/O operation. All subsequent timing is relative to this cue. As shown by the Motor (Tape) Speed curve, this pulse turns on the Tape Recorder. However, since it takes a finite amount of time for the tape to get up to speed, no I/O operations can be performed reliably until after a delay interval. For output the delay is TOD, as shown. For input operations, the device begins to listen for data prior to the time when the first bit is expected. The input delay TID is thus shorter than the output delay TOD. This strategy guarantees that the unit is always listening to the tape before the first actual input data bit comes along --- so that no data will be lost while reading. Somewhat arbitrarily, the input delay was set at TID=2 seconds in the prototype and the output delay was set at TOD=3 seconds. The TID delay is more than sufficient to allow for the motor start transient.

Note that with this scheme of recording with self-clocking data, the input operation is always re-synchronized to the tape at each block boundary, so that any synchronization errors (eg: missed bits) will be confined to the block in which they occur, and minor differences in block timing due to motor peculiarities will not be additive over the entire tape cassette.

1.2.3 MANUAL OPERATIONS:

This device requires manual intervention in order to operate in an orderly and well-defined manner. The need for manual operator intervention is a result of efforts to keep the price down: there is a trade-off between electromechanical controls and price. Thus in order to perform an input or output pass through the tape cassette, the following series of manual preparations must be performed:

1. Place the MOTOR DRIVE switch (S2) into its manual position (β in figure #6a) to provide power to the recorder.
2. Mount the desired data cassette in the recorder and rewind it to the beginning. One advantage of cassette media versus reel-to-reel is that there is a well defined beginning of tape position: the stall point in a rewind.
3. Place the MOTOR DRIVE switch (S2) into its computer control position (γ in figure #6a). In this position the state of the motor is governed by logic circuitry, and should be initially off provided the Tape Control module and other elements (hardware and software) are properly initialized.
4. Pick an input or output mode of operation depending upon the purpose of the pass through the cassette. Do this by setting the Input/Output mode switch and by setting the tape recorder controls accordingly:
 - Set the recorder to RECORD if an output operation is to be performed, and the I/O Mode is set to output.
 - Set the recorder to PLAYBACK if an input operation is desired and the I/O Mode is set to input.

Note that in either case, the tape recorder power is off so that "starting" the machine by means of its controls will have no immediate effect. Later, the logic circuitry will activate the recorder at the appropriate time.

5. Prior to the first I/O operation after the above setup, the cassette must be initialized. This is accomplished by turning on the recorder for a time period set by a one-shot which is fired by switch S2 (TAPE INITIALIZE). The nominal 5-second period of this operation is sufficient to advance the tape beyond its non-magnetic plastic leader.

After these setup operations are completed an arbitrary number of I/O operations can be performed on the tape provided that manual intervention does not move the tape physically. One block of data is written or read in each such operation with a length set by the value presented to the Data Count inputs (socket -24-) at the start of the operation.

With manual information to repeat step 4 it is possible to change from input to output mode and vice versa for successive blocks on a single pass of the tape; however the previous I/O operation must have been completed and care must be taken to avoid moving the tape while changing the mode. If such mode switching is contemplated appropriate programs can be written to present instructions on data displays and to recover from errors induced by inadvertently moving the tape. Much will be said on such topics in future articles of this series.

1.3 SUBSYSTEM BLOCK DIAGRAM:

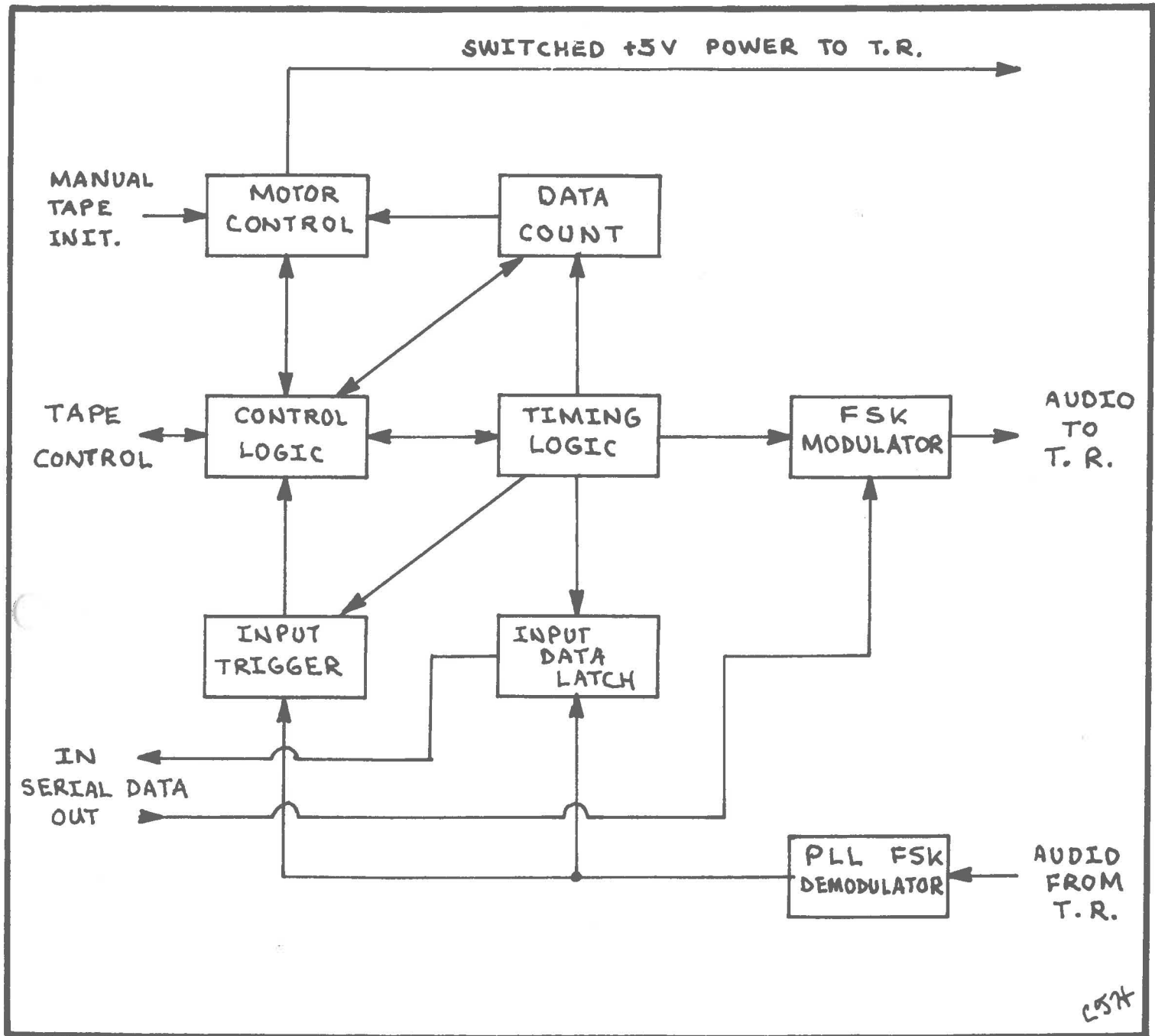


Figure 5. Subsystem Block Diagram

Figure #5 shows the subsystem block diagram of the Audio Cassette Mass Storage System. This figure outlines the major functional divisions of this I/O device. Details of the entire circuit are found in a later figure, #6a and #6b, and discussed in section 1.4 below. The major functional divisions of the design are:

- TIMING LOGIC: Central to the whole concept of this unit is the timing logic used to sequence operations. This section uses a high frequency clock which is divided to get the frequencies of f and $2f$ used for data modulation, and further divided to get the 15-parts of each data bit period (at a frequency of $f/30$).

- DATA COUNT: A 16-bit counter is incorporated in the design to provide a count of the number of bits to be transferred. Up to 65,536 bits can be read or written in one operation. This count is loaded from a fixed (hardwired) value or is under control of the Tape Control module, depending upon what is plugged into socket position -24-.

- CONTROL LOGIC: This section operates directly from inputs provided by the Tape Controller, Data Count and the Input Trigger. The state of the subsystem is determined by various gates and time delay elements incorporated into this section.

- MOTOR CONTROLS: This section contains a motor state latch and the transistor switching circuit which turns on the tape recorder power supply. The tape initialization timer and a separate floating power supply for the tape recorder are part of this section.

- FSK MODULATOR: The logic contained in this section consists of two flip-flops set by timing logic to create the control signals for the Bit Mark and Data phases, as well the FSK switching logic which generates the output audio by choosing f or $2f$ depending upon phase and data.

- PLL FSK DEMODULATOR: This block contains the Phase Lock Loop (PLL) which converts FSK audio inputs back into a voltage proportional to frequency. A comparator generates a logic signal (raw data) from the PLL output, which is then inputted to the Trigger and Data Logic.

- INPUT TRIGGER and DATA LATCH: These blocks have a common gated Schmidt Trigger input taken from the demodulator. The Trigger Section, working in conjunction with various control logic and timing elements, defines a pulse corresponding to the f - $2f$ transition of the input signal, synchronizing the system to the tape data. The Data Latch retains the data as sampled during the input data phase, so that the Tape Control unit can read that data with the Data Clock which occurs later during the Null Phase.

1.4 DETAILED DESIGN:

Following a few comments about the conventions used in the logic diagrams of figures #6a and #6b, this section covers the detailed operation of the circuit. References should be made to both the logic diagrams and the system concepts presented above in order to understand the design with the aid of these notes.

1.4.1 LOGIC DIAGRAM CONVENTIONS:

Figures #6a and #6b show the complete logic and circuit diagrams of the Audio Cassete Mass Storage System. Due to the size of the circuit it has been divided into the two parts shown. Internal connections between the two sections of the diagram are indicated by the notation of a single capital letter and the symbol . External connections via the interface socket (socket -25-) are denoted by a number next to a dashed line indicating the pin number. A second socket, -24- is used to interface the 16-pins of Data Count input, but is not noted explicitly in the diagrams.

With the exception of logic inverters, all IC package numbering is found inside the symbol for the component in question, denoted by a number preceded and followed by a hyphen, eg: -11- stands for IC socket position number 11. For the MSI functions represented by rectangular boxes, internal logic designations of pins are noted within the boxes and external pinouts are noted outside the box. For gates and inverters only the external pin numbers are shown since function follows from the shape of the symbols used. In the text of this article, the notation "-11.7-" is used to indicate a reference to the pin of an integrated circuit socket, in this instance pin 7 of socket position 11.

1.4.2 TIMING LOGIC DETAILS:

PHASE STATE: The Phase State is defined by the contents of the 7493 counter -14- at any given time. The position of the output execution within a bit period is defined by the 15-successive states of this counter. Connected to the counter is a Phase Decode block consisting of a 74154 selector -15- which defines 15 negative-logic (low state = logic 1) lines, only one of which is selected at any given state of the inputs. The outputs of -15- are used as follows:

- State 1 (pin -15.2-) defines the beginning of the Bit Mark phase for output.
- State 6 (pin -15.7-) defines the end of Bit Mark and the beginning of the Data phase during output.
- State 9 (pin -15.10-) defines the data sampling clock for input operations. When this state is present, the Data Latch comprised of -16c- and -16d- is reset, after which the current input data will define the new state of the latch.
- State D (pin -15.15-) defines the Data Clock pulse and resets the Trigger Latch for input operations.
- State E (pin -15.16-) defines the Wait State used for input synchronization to data. This is the last state to occur in an input bit period, and is not used during output.
- State F (pin -15.17-) is a null state used to reset the Phase State counter and achieve a divide-by-15 operation during output operations; during input this state is never reached.

CLOCK GENERATION: An oscillator formed by -19a-, -19b-, and -19c- generates the basic timing frequency of 24,000 cycles per second used for all digital clocking in this device. This frequency is noted as "8f" in the diagram. The potentiometer R10 controls the frequency of oscillation over a fairly wide range, and ulti-

mately determines the data rate of the device. The $2f$, f and $f/2$ clock signals required by the system are generated from the $8f$ signal of the oscillator using a 4-bit divider, a 7493 circuit in socket position -21-. This counter shares a common reset with the Phase State counter (PHASE RESET in the drawings) in order to guarantee that all timing logic will be reset to a unique and well defined state when necessary (ie: all zeros in -21- and -14-). The $f/2$ output of -21.11- is the source of the PHASE CLOCK signal gated to the Phase State counter by control logic elements.

1.4.3 DATA COUNTER LOGIC DETAILS:

The Data Counter shown in this design is a 16-bit synchronous counter made up of four 74193 circuits mounted in socket positions -2-, -3-, -4- and -5-. The parallel load inputs of all 16-bit positions of this counter are shown as open circles in figure #6a. Bits 0 through 15 are understood to be connected to pins 1 to 16 respectively of the Data Count input socket -24-. By means of this socket these input values can be wired up to a DIP header plug with a fixed number pattern, or they may be connected via that type of plug to the outputs of registers in the Tape Control unit. In either case, the information presented at this plug (-24-) is loaded into the counter once per I/O operation when the I/O Start pulse is received. The counter is then decremented once for each bit in the transfer by the Data Clock pulse. When the last stage (high order counter -2-) underflows the Motor State Latch is reset and the I/O operation terminates.

1.4.4 CONTROL LOGIC DETAILS:

The gates and logic blocks in the lower part of Figure #6a comprise the Control Logic elements of this unit. Identifying by integrated circuit socket position, the following notes describe the logic in some detail:

-6- is the output start delay timer. Its main purpose is to inhibit the operation of the modulator until the tape has settled down during an output operation. -12a- forms the logical product (AND) of the output delay with the state selection data, so that the OUTPUT-INHIBIT is only active if output has been selected.

-7- is the input start delay timer. It serves the same purpose for input operations which -6- performs for output, but has a shorter time period so that input "listening" begins prior to the first bit on the recorded medium. Its output is inverted and used to set an input clock-inhibit latch formed by -8c- and -8d-. The inverted form of this signal is also used to inhibit PHASE RESET during setups at the beginning of an input operation. In the output mode the operation of this logic is ignored.

-13c- defines the GATED PHASE CLOCK input to the Phase State counter -14-. When enabled by a lack of input or output inhibit signals the PHASE CLOCK is inverted and passed to the Phase State counter. The inhibiting conditions are:

For output, the output time delay of -6- gated by -12a-.
For input, count state E (-15.16-) gated via -11c- and -12b-.

In the input mode when the state count reaches state E (binary "1110") and halts, the next input trigger pulse generated from the input resets the state and allows the clock to run again. This is the Wait state mentioned above which is used to synchronize the I/O electronics to the actual bit period thereby immunizing the system against errors of an imprecise tape mechanism.

-13b- defines the PHASE-RESET signal as a logical sum (or) of three signals, using a NAND operating on negative logic sources:

1. State F (-15.17-) is used to cause the Phase State counter to have a 15-state cycle by resetting after the fifteenth state (state E, -15.16-).
2. IO-START is also used to reset the counters, thus defining a unique initial state at the start of all I/O.
3. IN-TRIG is a signal generated from the input trigger pulse by -13a- when in the input mode after the delay at the start of an operation is over. In the input mode, the pulse gated via this line causes a transition from the Wait state to state 0 at the start of a bit period.

-9b- is a NOR used as a negative logic AND function in order to gate the input trigger to the PHASE-RESET logic above, as well as the input clock inhibit logic. A pulse will be defined and pass this gate only if the Phase State count is waiting in state E as described above.

-9d- is used as a positive logic AND function to generate the Data Clock signal unless inhibited directly by the input clock inhibit latch (see discussion of -7-). Note that for output data there is no direct and explicit data clock inhibit term. The Data Clock is inhibited during output start-up by forcing Phase State counter reset until the output delay is completed. In both input and output modes, the Data Clock is inhibited after the end of an I/O operation when the Motor On signal is reset.

1.4.5 MOTOR CONTROL DETAILS:

The TAPE INITIALIZE CIRCUIT is composed of the time delay circuit -1- and a pushbutton switch S1. The value of R1 is adjusted to give a nominal tape-leader length delay of 5 seconds — sufficient to space the cassette forward past the plastic leader strip. The output of this circuit shares control of the tape recorder power function via the logical sum formed in -9c-.

MOTOR STATE LATCH: two NAND sections, -8a- and -8b- form a set-reset flipflop which governs the state of the tape recorder power during normal operation. This latch is set by the I/O Start pulse (interface pin 10) and is reset by the end of data count-down when an underflow pulse is generated from -2.13-. This bit also controls the modulator flip-flops to inhibit data generation

and clocking after the end of the data block on output. (See figure #6b, pins -15.18- and -15.19- for this usage.)

The TAPE RECORDER DRIVE BOX is a separate module built into a small plastic case which may be kept near the recorder. This box is separated from the rest of the logic diagram by means of an appropriate dotted line in figure #6a. This section of the diagram houses the tape recorder drive circuitry, a power indicator lamp and a separate power supply for the recorder. The control signal from the main subsystem logic consists of two interface plug pins: pin 8 is ground and pin 15 is the logic signal to drive the recorder. In addition, this box includes the MOTOR DRIVE mode switch (S2) and the LED indicator L1 which is in parallel with the tape recorder power supply. The tape recorder is represented in the drawing as a coil to indicate the inductive nature of the load it places on the driver. The protection diode D1 is used to guard against transistor damage due to inductive back EMF in the motor coils. A two-transistor buffer switches the tape power supply under logic control. This unit can be built and debugged separately — and can incidentally serve as a battery replacement when the recorder is not used for computer purposes.

1.4.6 MODULATOR DETAILS:

The FSK MODULATOR consists of the logic found in the upper right corner of figure #6b. Two set-reset flipflops form the Bit Mark and Data phase signals used to control the FSK switch. In the Bit Mark phase gate -20a- always has a logical 0 presented to pin -20.1- so the NAND output -20.3- always has a logical 1 value. This results in the unconditional choice of the 2f signal at the FSK Switch as is desired during Bit Mark. In the Data phase, the situation changes with -18c- acting as an AND gate to enable the data input to pass to the modulation switch via -20a-. Following the Data Phase, since neither modulator flip-flop is set, both inputs of -20a- are logical 1 and the modulator switch is thus unconditionally in the Null state configuration outputting frequency f. Note that in the beginning of an output operation, OUTPUT INHIBIT is used to reset both flip-flops unconditionally until the time delay is up. This guarantees that a Null phase leader will be recorded continuously prior to the first data bit.

1.4.7 DEMODULATOR DETAILS:

The demodulator logic of this system consists of the Phase Locked Loop -23-, the comparator circuit -22- and associated discrete components. The audio signal from the speaker terminals of the tape recorder is input to the PLL through a resistor and the diode clipping network of D2 and D3. The clipper protects the 565 PLL against excessive signal by limiting signal voltage to the diode forward voltage drop. Potentiometer R11 is used to adjust the free running frequency. The output of the PLL is a voltage proportional to the input frequency detected. This voltage and a reference voltage are both fed from the PLL to the comparator via a filter (R12 and C9). The comparator in turn produces a logic signal which is then cleaned up

and made TTL-compatible by the Schmidt trigger input of the first gate stage it drives. The two Zener diodes Z1 and Z2 are used to define the +6.3 and -6.3 power supplies for the PLL and comparator from the +11 and -11 inputs shown.

1.4.8 INPUT TRIGGER & DATA LATCH DETAILS:

The input trigger logic consists of the Schmidt trigger gate -17- and a set-reset latch formed by -16a- and -16b-. (The Schmidt trigger output also drives the Data Latch.) During the end of a bit period, the f signal is present on input and produces a low level logic signal at the comparator output. This guarantees that during the Null phase of input the output pin of the Schmidt gate (-17.6-) will be high. When the Trigger set (State D, -15.15-) is reached this forces the Trigger latch output (-16.3-) into the logical 1 state. Tracing through the control logic of -9b-, -13a- and -13b- shows that the Phase Reset line will be zero when the input "Wait" state (State E, -15.16-) is reached, and the system will be quiescent. As soon as the f-2f transition occurs however, the system state will change as follows: the output of -17.6- will drop to logical 0 and thus reset the Trigger Latch producing a logical 0 signal at -16.3-. This level will then propagate through -9b-, -13a- and -13b- to the Phase State counter's PHASE RESET line. This resets the Phase State, which is propagated through the Phase Decode logic of -15- and removes the State E output (while enabling the State 0 output). The Phase Clock is no longer inhibited with the end of State E, and the trigger signal is cut off when State E is no longer available to enable the AND logic of -9b-. The output of -9b- is thus a pulse whose width is set by the sum of the propagation delays of -13a-, -13b-, -14- (reset to output), -15- and -9b-. Using nominal figures from IC specifications this sum is approximately 74 nanoseconds.

2. CONSTRUCTION AND TESTING:

2.1 TABLES:

To aid in assembling and testing your own version of this I/O unit, several tables are included at the end of this plan in addition to the complete logic diagram of figures #6a and #6b. These tables are the following:

Table I: Package Summary. This table contains a summary of all integrated circuit and other socket positions in the circuit, showing number of pins, power and ground connections and other information.

Table II: Other Electronic Parts. This table lists all the miscellaneous parts used in the design.

Table III: Interface Sockets. Two sockets, -24- and -25- are used to interface this circuit to the world of the Tape Control module. The complete list of signals wired to these sockets is summarized in this table.

2.2 NOTES ON CONSTRUCTION TECHNIQUES:

Due to the complexity of this circuit it is strongly recommended that the solderless wrapped wire method of interconnection be employed in the construction of this design and all the plans in the Experimenter's Computer System series. For those individuals who are familiar with the technique, no introduction is necessary; for individuals who are not familiar with wire wrap interconnection, the publication entitled "Solderless I.C. Prototyping Techniques" (M.P. Publishing Co. Number 73-1) is available and provides information needed to utilize this method of wiring.

The main logic board of the system should be fabricated as a single module using copper-clad board to provide a good ground plane. The entire circuit (excluding the Tape Recorder Drive box) can be laid out on a single piece of blank P.C. board or Vector stock measuring 4" by 8". Since the entire Experimenter's Computer System will involve many boards with interconnections, it is recommended that a card cage system be employed, in which case the dimensions used for this board will have to be consistent with the card cage used. The discrete components of the main logic board for the Audio Cassette Mass Storage System are mounted on insulated standoff terminals in the prototype. This includes the timing capacitors and resistors of the oscillator, demodulator and delay oneshots. The trimming potentiometers were simply glued to the circuit board with a small amount of contact cement.

Wiring should begin with the power and ground connections of all the dual in line sockets (summarized in Table I.) Following power wiring, methodically connect all signal wiring as in the diagram of figures #6a and #6b. One of the best ways to ensure that each connection is covered is to highlight the circuit diagram line for the connection in red ink (or other color) after it has been wired and inspected. It is suggested that a wire list be made as well if more than one copy of the unit is contemplated. Extra copies of the logic diagram page of this publication may be purchased from the publisher for \$.50 in single quantities to replace your original diagram if you make the suggested markings during construction.

In laying out the board, make provision for "decoupling" capacitors at several points in the wiring of the +5 volt power bus for the TTL circuits. With any TTL circuit noise can and will be propagated along the power lines due to switching transients --- and can randomly affect operation. Good construction practice thus includes placing several sets of decoupling capacitors around the power bus to store energy for the TTL switching process. For this circuit, three 10 mfd electrolytics in parallel with .1 mfd ceramic capacitors will serve this function when wired from three different locations on the power bus to ground.

2.3 TESTING THE CIRCUIT:

2.3.1 COMPONENTS:

As the builder of this design, you are interested in making it work and correcting your own wiring errors, not in testing out the components you buy. It is extremely important with a complex system such as this

one that you be able to rely upon the quality of the logic components you buy. Accordingly, never under any circumstances buy "hobby" quality circuits unless you are prepared to become frustrated by circuit components which must themselves be tested before use. Even with first line circuits purchased from a distributor or surplus dealer, occasionally you will find electrical problems with the IC components. In the process of testing, by proceeding one step at a time with a healthy skepticism of component quality, most such bad components can be isolated.

2.3.2 INTERCONNECTIONS FOR TESTING:

At this stage in the development of the Experimenter's Computer System, the only way to test out the Audio Cassette Mass Storage System is manually. Later in the series, articles devoted to diagnostic programming for this device and other devices will be appearing — techniques which provide for more comprehensive and thorough testing. For the purpose of manual testing, the DIP header plugs for socket positions -24- and -25- must be prepared to give access to the system's interfaces. For independent testing, thus two dummy plugs are needed:

1. DATA COUNT: The data count interface socket -24- must be set up with a bit pattern supplying a data count for testing. Unconnected pins can be assumed to be logical "1" for purposes of this testing, and pins connected to ground are logical "0". For purposes of testing, wire pins 1 to 7 together on the plug and connect them to a clip lead. A maximum length count is achieved by leaving the clip lead dangling; a count of binary "0000000111111111" (511) is achieved by connecting the clip lead to ground. At 100 baud the maximum length count will run for 655.35 seconds and the shorter count will run for 5.11 seconds. Other count values can be set by wiring this plug differently.

2. INTERFACE SOCKET: The general interface socket -25- must be provided with a plug connected to wires which are stripped at one end and connected to socket pins via solder at the other end. Clip leads can then be attached to the stripped ends of all 16 wires when needed during testing. In wiring this plug, all wires should be labelled with tags of masking tape to identify pin number and interface function as found in Table III. This will help eliminate the probability of confusion during testing.

2.3.3 ORDER OF TESTING:

One way of speeding up the process of testing a circuit such as this, of speedily finding wiring and component errors, and of using your time to best advantage is to use an orderly and methodical approach to testing. The following is an outline of the sequence of testing needed to manually verify the operation of this circuit. The theory of operation as described above and the logic diagrams of figures #6a and #6b should both be consulted for detailed information used at each step.

- () 1. POWER WIRING: Verify and check out all power supply wiring before ever plugging in a single integrated circuit. It is not an absolute certainty that the circuits will go up in smoke with bad power connections, but such has been known to happen.

- () 2. CLOCK GENERATION: Verify operation of the system clocks as the first operational test after inserting integrated circuits. Without an operating clock, the system is dead. Set the oscillator frequency to 24,000 cps (8f) and verify 12Kc (4f at -21.12-), 6Kc (2f at -21.9-), 3Kc (f at -21.8-) and 1.5Kc (f/2 at -21.11-). Note that the system must be in output mode (interface pin 3 grounded) for these first few tests.

- () 3. TIME DELAYS: Check out the time delay circuits, socket locations -1-, -6- and -7- next. The following table summarizes the nominal adjustments:

Delay Circuit		Trigger	Pulse Width
()	-1-	Interface Pin 14	5 Seconds
()	-6-	Interface Pin 10	3 Seconds
()	-7-	Interface Pin 10	2 Seconds

- () 4. MOTOR DRIVE: Check out operation of the Tape Recorder Drive Box next. First, verify operation in the manual mode: the indicator LED (L1) should be on and the Tape Recorder should operate normally when its various buttons are pushed. Then, verify logic control of the recorder by placing S2 in the computer mode (X) and pressing S1 to trigger a tape initialization pulse via interface pin 14. This should illuminate the lamp L1 for 5 seconds, and --- if the recorder is in an operating mode --- turn on the motor for 5 seconds.
- () 5. OUTPUT MODE: Next verify operation of the circuit in its output mode of operation. Ground the wire connected to Interface Pin 3 via the test plug, thus setting the circuit into output mode. To initialize output operations, momentarily connect interface pin 10 to ground, imitating an I/O Start command. (If this proves unreliable it may be necessary to wire a pullup resistor of 1K ohms to the +5 volt supply to this pin using clip leads.)

The timing diagram of figure #3 can be produced on an oscilloscope with multiple channels and a chopped mode of operation. The scope trigger should be connected to -15.1- via a temporary wrapped connection or a test connector. Temporary connection of -5.3- (the low order data count bit) to the TTL Data Input interface, pin 11 will establish the alternate 1 and 0 data shown in figure #3. To observe Φ_1 (Bit Mark) connect a scope channel to -18.6-; to observe Φ_2 (Data) connect a scope channel to -18.8-; to observe the Frequency control trace, connect a scope channel to -20.3-; to observe the Input Sample clock connect a scope channel to -15.9- and to observe the Data Clock, connect a scope channel to interface pin 7.

As a final check look at the actual FSK output (interface pin 4) retaining the same scope trigger source. The audio signal should be observed to switch frequencies at each transition of the Frequency Control signal, -20.3-.

If it is desired to check the output block format by producing the timing diagram of figure #4 the use of a multi-channel storage oscilloscope is necessary, with a very low sweep rate. This display is interesting from a tutorial standpoint, but is not necessary for checkout purposes.

- () 6. PREPARING A TEST TAPE: In order to test the operation of the system with input data, it is necessary to create test tapes using the output mode of operation. Retain the setups of step 5 so that an alternating 1/0/1/0/1... etc. sequence will be written onto the tape. Run through a cycle of several blocks beginning with the manual initialization of the tape (see 1.2.3 on page 5) and proceeding by initiating new write operations after the preceding operation is done. For initial testing of the demodulator signal circuitry, use the maximum length block count to record about five minutes worth of test data in each block. A second cassette can be prepared to test out input control logic using the short block length suggested in section 2.3.2 above.
- () 7. DEMODULATOR TESTING: The first step in testing input is to test the Phase Lock Loop demodulator and adjust it for best response. For this portion of the input testing, the maximum-length block is desirable as prepared above on one cassette. If short blocks are used, by the time you get around to looking at the PLL output on the scope you find that the block has finished --- a rather unproductive situation!

Reinitialize the tape as in section 1.2.3, change the mode to input by removing the ground connection of interface pin 3, set the maximum length value into the Data Count by removing the dummy data count plug ground connection, and give an I/O Start cue by momentarily grounding interface pin 10 as before. The tape recorder should now start playing, and after the three seconds of leader delay, data should start coming into the PLL input via the FSK Data Input of interface pins 12 and 13. Check this input of audio signal by probing pin -23.2- with the scope. Note that there will be considerable "wow and flutter" audible in the signal if a cheap tape recorder is used, and that this will also show up on the scope trace.

Adjust the Phase Lock Loop free running frequency setting dynamically while observing the PLL output on pin -23.7-. Make initial adjustments of R11 with the maximum volume setting of the tape recorder output, until the "cleanest" output signal is obtained (the closest approximation of a square-edged logic-like signal.) To avoid finding false locking points the free-running frequency may be set prior to this adjustment by removing input and observing the frequency on pin -23.4- which should be approximately 4-4.5 Kc. After a large-signal lock has been established, decrease the amplitude of the tape recorder audio until the PLL output begins to break up. Increase the amplitude again slightly from this point, then readjust the free running frequency with R11 for the best PLL output "square-waviness". This procedure establishes the best center frequency.

- () 8. INPUT OPERATION: With the demodulator producing clean output signals from the decoded FSK signal, the next step is to test the logical operation of the entire system in input mode. For this purpose, the test tape prepared with short blocks should be employed instead of the long block cassette. Manually initialize the tape unit as usual, and cue an input operation with interface pin 10. Observe various logic signals in the system as before, using state 0 as the scope trigger. An initial indication of proper operation is for the system to operate on one block of data and then quit (assuming the short data count is presented at socket -24-.)

Using manual operations it is impossible to get more than a heuristic impression of the correctness of the input data. The circuit shown below as figure #7 uses an 8-bit counter to sum up the difference in the number of 1's and 0's read by the unit. While this is not a guarantee against compensating errors, the probability is that for short blocks one type of error will dominate (eg: zeros interpreted as ones or vice versa.) Using the interface pins as output from the tape device, prepare the tester by pressing the Reset button. Then initiate the tape I/O using the short data block prepared previously. At the end of the block, providing the alternating data was recorded and re-read properly with no compensating errors, the value in the counter display will be binary 01111111. This can be proven by noting that the initial condition is 10000000 and that there is one more zero on the tape block than ones, so the last bit read will shift the result down by one count. (511 alternating bits outputted with an initial state of 0 in the data count means 256 zeros and 255 ones.) After verifying short blocks, the input of maximum length blocks can also be tested with the same unit. In either case, the result in the counter is the difference in the number of zeros and ones actually read, with the overflow/underflow lamp indicating whether a gross imbalance was detected. The parts shown in this diagram are not included in parts lists. This device may be put together in short order to perform the indicated tests.

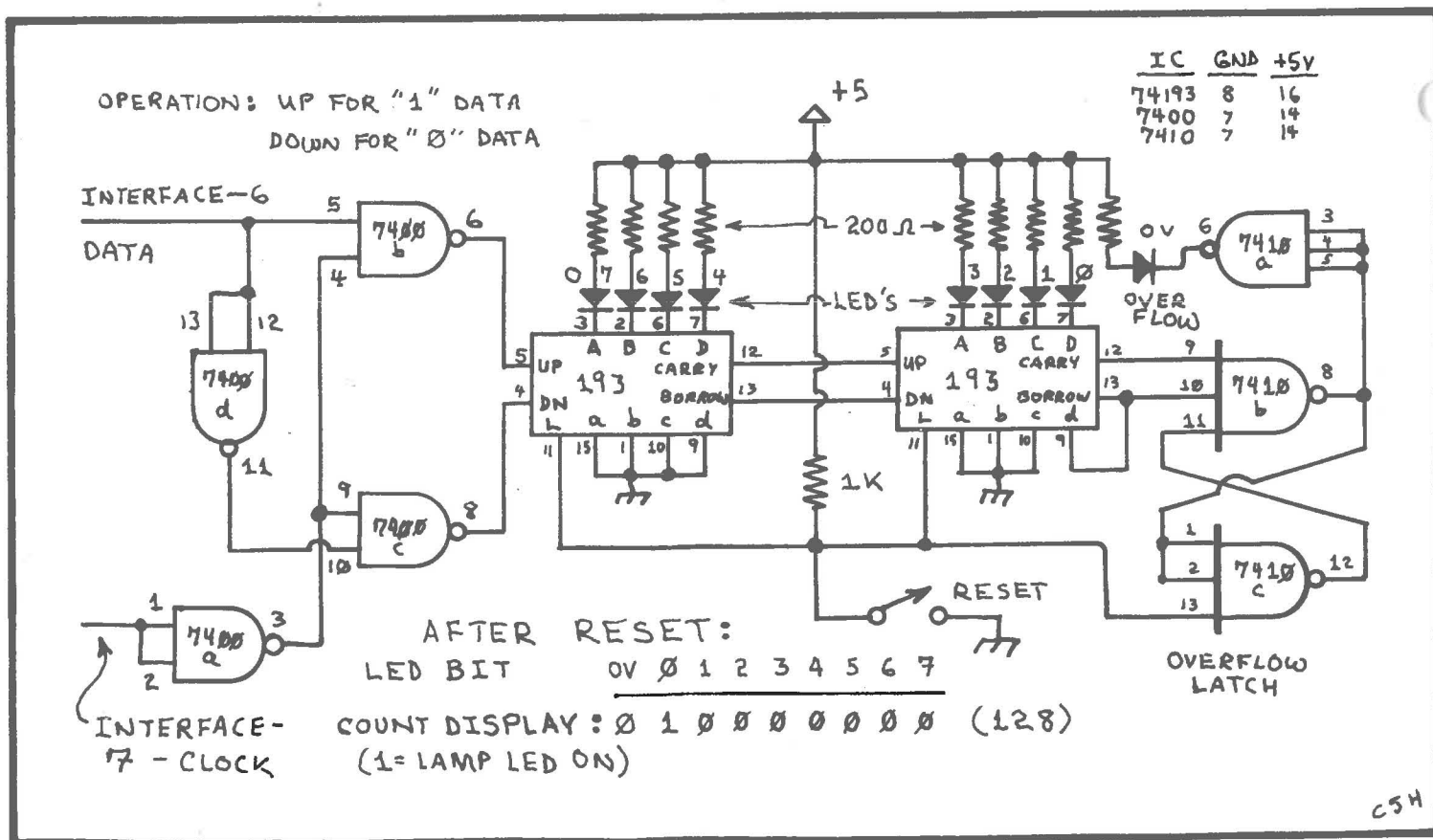


Figure 7. Uniform Data Test Unit

3. EXTENSIONS AND MODIFICATIONS OF THE SYSTEM:

The basic circuit as described in this plan was built and completely debugged by the author. As is the case with all engineering systems it should be understood that this not necessarily the only way in which to generate and record data, and that the parameters of this particular design can be adjusted further. Within limits it is possible to increase the performance of the basic design; for use within other systems contexts, portions of the control and data count logic might be changed. Some comments on the subject of modification are recorded here:

1. SIMPLE BIT RATE INCREASES: The "sure" way to achieve an increased data rate is to up the clock frequency and use as much of the tape recorder bandwidth as is possible. With a cheap imported cassette recorder, a "2f" signal of 10Kc may be possible in which case using the same frequency ratios a rate of 166 baud for data would be obtained. Such a move on a \$30 cassette recorder might be marginal, and should be statistically checked out with a suitable computer driven bit error checking program. If a reel to reel recorder, higher bandwidths are possible. A "2f" frequency of 20Kc may be possible on a quality audio recorder in which case the data rate could be doubled once more to 333 baud. Whenever a new frequency is chosen, the following cautions obtain: the PLL timing components will have to be changed, the oscillator timing capacitor may be outside the necessary tuning range, and no tape recordings made at lower data rates will be readable. To reiterate, the limiting factors on this sort of data rate increase are tape recorder bandwidth and the quality of the tape mechanism. The parameters of the 100 baud specification in this article are "safe" values which will most likely work on virtually any tape recorder.

2. BIT PERIOD FORMAT MODIFICATIONS: A second alternative to increase the data rate is possible if the mechanical noise of the tape drive is not a major source of frequency perturbations on input signals: the tape format can be altered. The circuit in figures #6a and #6b measures out precisely 20 cycles of the high frequency signal in $1/3$ bit period, and 10 cycles of the low frequency signal in $1/3$ bit period. Both these amounts exceed the minimum number of cycles required for the PLL to lock up. By changing the ratio of phase lengths to 2:3:3, with 8 cycles of 2f in the Bit Mark plus potentially 6 cycles of f in the remaining two phases, the same data can be potentially packed into $8/15$ of the time required for one bit in the present design. This set of ratios can be obtained by using the frequency f as Phase Clock and rewiring the outputs of -15- for the new ratios: State 1 starts Bit Mark, State 5 ends Bit Mark and starts Data, and State B ends Data phase to start the Null phase. By simply rewiring, this will increase data rate to 188 baud assuming the same clocks. Performing this modification in combination with that suggested above yields a potential upper limit of 625 baud on a tape recorder which can record a 20Kc signal.

3. DATA COUNT: The number of stages in the Data Count of this device is arbitrary and set for convenience in the Tape Controller. With control logic modification this counter may even be omitted if some other "end" signal inherent in data is provided by the computer.

Table I: Package Summary

#		Description	Pins	+5 Volts	Ground
1	1555	Tape Initialize	8	8,4	1
ord 5 2	74193	Data Count 4096's	16	16	8,14
3	74193	Data Count 256's	16	16	8,14
4	74193	Data Count 16's	16	16	8,14
5	74193	Data Count Units	16	16	8,14
6	1555	Output Start Delay*	8	8,4	1
7	1555	Input Start Delay*	8	8,4	1
8	7400	Motor & Input Latches**	14	14	7
9	7402	NOR Gates	14	14	7
ord 2 10	LM309K	Tape Power Regulator	3 Terminals, T0-3		
11	7404	Inverters	14	14	7
12	7400	NAND Gates	14	14	7
13	7410	NAND Gates	14	14	7
ord 5 14	7493	Phase State counter	14	5	10
15	74154	Phase Decode	24	24	12
16	7400	Trigger & Data latches**	14	14	7
ord 1 17	7413	Schmidt Trigger NAND	14	14	7
18	7410	NAND Gates	14	14	7
19	7404	Inverters (oscillator)	14	14	7
20	7400	NAND FSK Switch Gates	14	14	7
o 21	7493	Clock Frequency Division	14	5	10
ord 2 22	710	Input Comparator	14	Not Ap.	2
ord 2 23	565	Phase Lock Loop	14	Not Ap.	3
24	DATA COUNT INPUT INTERFACE		16	Not Applicable	
25	INTERFACE SOCKET		16	16	8

* -6- and -7- may be combined in one socket position as two 555's or as a single 556.

** These packages may be combined as a single 74279 package.

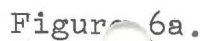
Table II: Other Electronic Parts

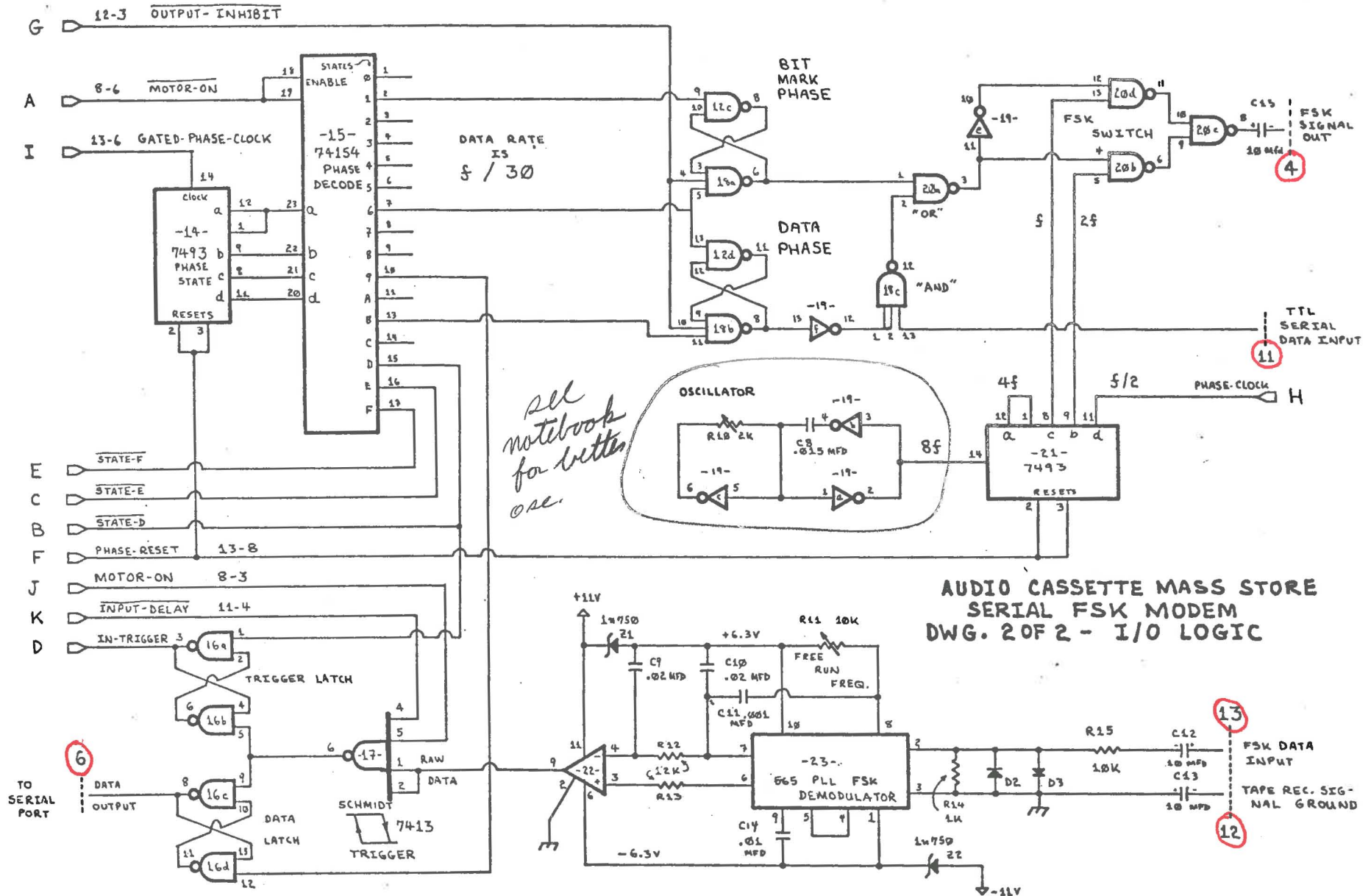
C1	10mfd 10v electrolytic	ord 10	R1	500K trim pot	ord 5
C2	.01 mfd ceramic		R2	1000	
C3	.01 mfd ceramic		R3	500K trim pot	
C4	.01 mfd ceramic		R4	500K trim pot	
C5	10mfd 10v electrolytic		R5	1000	
C6	10mfd 10v electrolytic		R6	1000	
C7	1500mfd min, 10v electrolytic		R7	100	
C8	.015 mfd	ord 5	R8	100	
C9	.02mfd	ord 5	R9	220	
C10	.02 mfd		R10	2000 trim pot	
C11	.001 mfd ceramic	ord 5	R11	10K trim pot	
C12	1 mfd 10v electrolytic	ord 5	R12	12K	
C13	1 mfd 10v electrolytic		R13	12K	
C14	.01 mfd ceramic		R14	1000	
C15	10 mfd 10v electrolytic		R15	10K	
D1	Silicon Switching Diode		S1	SPST Pushbutton	
D2	Silicon Switching Diode		S2	SPDT Center off toggle	
D3	Silicon Switching Diode				
L1	LED: 10 ma		T1	6.3v 1.2a Filament transformer.	
Q1	2n2222	ord	Z1, Z2	4.7v Zeners (1n750)	
Q2	2n5190				

Miscellaneous: Full wave bridge rectifier, line cord & plug, case and housing for Tape Recorder Drive Box, external power supply and auxiliary-input/speaker plugs for the Tape Recorder.

Table III: Interface Sockets

Socket -24-			Socket -25-	
Pin	Count if "1"	Wire To	Pin	Signal Description
1	32,768's	-2.9-	1	+11 volts power (Max +12v)
2	16,384's	-2.10-	2	Not used
3	8,192's	-2.1-	3	Input/Output Mode Select (Panel)
4	4,096's	-2.15-	4	FSK Audio Signal Out
5	2,048's	-3.9-	5	Not Used
6	1,024's	-3.10-	6	Serial Data to Controller
7	512's	-3.1-	7	Serial Data Clock
8	256's	-3.15-	8	Ground (Power & Signal)
9	128's	-4.9-	9	-11 volts power (Max -12 magnitude)
10	64's	-4.10-	10	I/O Start Line (Neg. Logic)
11	32's	-4.1-	11	Serial Data from Controller
12	16's	-4.15-	12	Tape Recorder Signal Ground
13	8's	-5.9-	13	FSK Audio Signal In
14	4's	-5.10-	14	Tape Initialize Pushbutton
15	2's	-5.1-	15	Tape Drive Signal
16	1's	-5.15-	16	+5 volts power.





PREVIEW OF COMING ATTRACTIONS

The Audio Cassette Mass Storage System as described here is but one component of the Experimenter's Computer System — the object of this series of articles. On the occasion of this first printing of #ECS-2 (April 1974) the following additional articles are scheduled for publication in the near future:

ECS-3: MICROCOMPUTER CPU, BOOTSTRAP & INTERRUPT LOGIC. This article describes the basic microcomputer CPU, the buffering needed to interface it to the outside world, the bus control concepts involved, bootstrap memory operation, and the design of interrupt logic allowing 8 levels of software-decoded priority.

ECS-4: 256-BYTE STATIC RAM. This article concerns the first form of Random Access Memory module to be employed in the Experimenter's Computer System. For initial programming and testing one or two pages of memory constructed according to this plan will suffice to demonstrate the operation of the computer system, its I/O capabilities and self-test programming. This form of memory is not the most economical in large quantities, but it does permit the incremental addition of memory. Accordingly, a later article in the series will discuss the more economical use of larger LSI RAM chips in bigger modules.

ECS-5: I/O PAGE DECODE LOGIC. One of the simplest forms of I/O to deal with conceptually is that of a dedicated area of the computer's memory address space which maps directly into real-world I/O operations: under this scheme all I/O reduces to memory transfer operation. This facility is provided in the Experimenter's Computer System by dedicating one 256-byte page of memory to I/O usage with a common control and address-decode mechanism. Each individual byte of that page is potentially an I/O device register used for data transfer or control purposes. A typical device will use two or three of these addresses grouped together for programming convenience.

ECS-6: TAPE CONTROLLER: This article describes the Tape Controller used to interface the Audio Cassette Mass Store of ECS-2 to the bus oriented logic of the Experimenter's Computer System mainframe. This subsystem operates under direct control of the I/O PAGE DECODE LOGIC described in ECS-5 and interfaces with the buffered data bus of ECS-3. It has primary responsibility for parallel/serial conversion of data, and the generation of periodic interrupt signals for the CPU during the course of tape I/O operations.

The Experimenter's Computer System: Part 4

2 5 6 - B Y T E R. A. M. P A G E

by Carl T. Helmers, Jr.

INTRODUCTION:

This article is the fourth number in the Experimenter's Computer System series. It continues the description of hardware begun in earlier articles with information on a standard module containing 256 bytes of memory, the smallest memory increment which can be conveniently added to the system. The article contains the following information:

1. Hardware description.
2. Summary Tables & Notes on Construction.
3. Programming Notes: Testing the Module.

The information found in this article must be supplemented by reference to the third article in the ECS series, #ECS-3: "Micro-Computer CPU & Bootstrap Logic.)

HARDWARE DESCRIPTION:

The center page of this article contains the detail logic diagram of the 256 byte memory page design. There are two primary divisions of the logic in this design:

1. The Memory Array logic consists of 8 Signetics 2501 IC packages (or the equivalent 1101) and an associated bus interface for data.
2. Control Logic consists of two 7485 binary comparators used to generate a page select signal, plus a set of NAND gates (one 7400 package) used to develop a data bus enable signal and a memory write pulse - both in terms of the general "WRITE-CLOCK" and "CPU-INPUT" signals provided by the ECS system's central processor design.

MEMORY ARRAY:

This is the first of several alternate designs for memory modules which will be presented in the course of this series of articles. As the first, one design criterion was to make the increment sufficiently inexpensive for experimenters of limited finances - hence the decision to limit the module to 256 bytes of memory. A logical choice for the memory circuit is the 2501 static RAM IC of Signetics manufacture, or the equivalent 1101 circuit produced by several other suppliers. This memory is presently available from surplus houses for prices in the \$2-3 range, so the total cost for the memory portion of the design (8 chips) will be in the \$16 to \$24 range depending on your supplier's prices. For the record, a current distributor price (October 1974) for these IC's is \$6.00 in unit quantities.

The memory array is located in the upper right hand region of the detail logic diagram, Figure #1. Each of the 8 integrated circuits in the array contains a "one bit slice" of the 8-bit words in the page. The addressing of the 8 chips is identical, and is derived from the low order 8 bits of the buffered address register maintained in the ECS CPU design. For clarity in drawing, the address lines are shown going from one memory chip to the next - where it is understood that a common connection will be made to identical pins of the 8 IC's for the array.

One point which should be discussed is the loading of the buffered address bus caused by common wiring of all 8 chips. The nominal TTL fanout of the address buffers in the CPU design (7437 circuits) is 30. In order to put off additional buffering as long as is possible as the system grows, a good design rule is to keep the loading at a minimum for each additional module of the system. What is the loading in this case? It turns out that a 2501 circuit - being MOS - represents a much smaller load than the ordinary TTL unit load. Using the worst case figures of the manufacturer's specifications, a 500 nanoampere input load current in the low state, eight 2501 inputs wired together would represent a total of 4000 na or 4.0 microamperes.... in the worst case. Since the typical TTL low state unit load is 1.6 ma, based on these considerations wiring the 2501's directly to the address lines represents a unit load of only $.004/1.6 = .0025$ unit loads. This discussion is fine for DC worst case - but there remains the consideration of dynamic effects. Each 2501 represents an effective capacitance of 10pf (per specs) on the line plus the total capacitance of all the extra wiring. As more and more units are wired to the address busses, a considerable capacitive loading of the buffer gates will result - a situation which depends in detail upon specific wiring lengths, layouts and interconnection techniques. This capacitive loading will tend to slow down the transitions of the address lines, as can be verified by experimenting with an oscilloscope, a 7437 gate package, a pulse generator and a capacitance value of perhaps .01 microfarads across the output of the 7437 section to ground. This .01 mfd (10,000 pf) represents what the 7437 might see when the number of address loads approaches 10 to 15 boards with both IC input capacitance and wiring grid capacitance effects totalled. After some consideration of this issue, it was decided to wire the 2501's directly and to treat the whole memory array as a single TTL unit load - opting for a conservative approach. This choice also keeps address loading for the low order bits the same as the single TTL loading represented by the wiring of the page selection comparators to the high order bits.

The output data is taken from the 2501 memory circuits via the complement pins - the D pin (14) - of each chip. The output data is connected to the bus via the 8T09 interface gates which invert the data. Thus by presenting the complementary form of the information to the interface gate inputs, the double inversion will result in the correct sense of data presented from the memory to the bus.

CONTROL LOGIC:

The control logic of the ECS-4 design is used to determine when this page of memory has been selected - and given its selection, to route the memory write signal and the bus enable signal to the appropriate users. The determination of page selection is performed by the MSI digital comparator circuit, the 7485.

The 7485 comparator can be used to perform a magnitude comparison if desired, however in this application its use is limited to a test for equality of two bit patterns. One of the two bit patterns is supplied by the 6 address jumper plug inputs used to determine the page address which is to be associated with this memory module. Two remaining 7485 inputs on the "B" side of the comparators are fixed-wired to a logical zero and logical "1" respectively.

The second bit pattern is provided by 6 bits of the high order ("H") portion of the CPU's demultiplexed address output, plus a single bit-input from the "master enable" signal. The eighth bit of the comparison's "A" input is fixed wired to ground corresponding to the ground (logical "0") input to that bit from the "B" side. If all seven bits of the address plus master enable input agree with the module address input and a desired "1" state for the master enable signal, then the output of the comparison, pin 6 of IC 4, will be a logical "1" signal. Otherwise, the "page select" line will be logical "0".

(The "master enable" signal is created in the CPU design of article ECS-3. It is used to over-ride all normal memory selection logic during an interrupt PCI cycle so that the interrupt "RST" instruction may be "jammed" onto the bus instead of the usual memory outputs. The sense of "master enable" is as follows: "1" indicates allow page selection; "0" indicates inhibit page selection.)

The output of the comparison logic is the "page select" line. This line is used to enable two logical product terms: 1) for PCI and PCR cycles in which memory output is read from the bus, the bus enable signal is formed by the product "page select" and "cpu-input" where the "cpu-input" term is a master bus control signal generated in the ECS-3 CPU design. 2) for PCW cycles in which the memory is written using CPU-generated data on the bus as input, the memory write pulse is formed by a logical product of "page select" and "write-clock" where the "write-clock" signal is generated in the CPU design of ECS-3. In the write pulse logic, two extra inversions are required to transform the signals into a usable form.

A NOTE ABOUT POWER:

The schematic of a simple zener diode network is shown at the upper left in the diagram. This network is used to generate a -9.1v bias for the 2501 chips in a manner similar to that used in the ECS-3 design.

TABLE I: Package Summary List...

DIP#	Iden.	Pins	Description	+5v	Ground	-9v
1	I/O#1	16	Data Bus & Miscellaneous	16	15	*
2	I/O#2	16	Address Inputs	-	-	-
3	7485	16	Page Address/Mstr En. Comp.	16	8	-
4	7485	16	Page Address Comparator	16	8	-
5	7400	14	Control Logic	14	7	-
6	8T09	14	Tristate Interface Gate	14	7	-
7	8T09	14	Tristate Interface Gate	14	7	-
8	Addr.	8	8-pin Address Jumper Plug	-	-	-
9	2501**	16	256x1 Memory Circuit, bit 0	5	-	4,8
10	2501**	16	256x1 Memory Circuit, bit 1	5	-	4,8
11	2501**	16	256x1 Memory Circuit, bit 2	5	-	4,8
12	2501**	16	256x1 Memory Circuit, bit 3	5	-	4,8
13	2501**	16	256x1 Memory Circuit, bit 4	5	-	4,8
14	2501**	16	256x1 Memory Circuit, bit 5	5	-	4,8
15	2501**	16	256x1 Memory Circuit, bit 6	5	-	4,8
16	2501**	16	256x1 Memory Circuit, bit 7	5	-	4,8

*-12 volts is routed via pin 14 to the zener network which generates the -9 volt bias for the memories.

**The 2501 circuit is functionally equivalent to the 1101 number manufactured by several companies. As an aside, with due care to pinout differences and loading factors, any 256x1 memory which is TTL compatible can be used with this basic design.

TABLE II: Interconnection Lists...

I/O Socket #1 - Bus & Miscellaneous

Pin 1 - Bus data bit 0 (D0)
Pin 2 - Bus data bit 1 (D1)
Pin 3 - Bus data bit 2 (D2)
Pin 4 - Bus data bit 3 (D3)
Pin 5 - Bus data bit 4 (D4)
Pin 6 - Bus data bit 5 (D5)
Pin 7 - Bus data bit 6 (D6)
Pin 8 - Bus data bit 7 (D7)
Pin 9 - NC
Pin 10 - CPU-Input Bus Control
Pin 11 - NC
Pin 12 - Master Eneable Input
Pin 13 - Write Clock (inverted)
Pin 14 - Power, -12 volts
Pin 15 - Power, ground
Pin 16 - Power, +5 volts

I/O Socket#2 - Addressing

Pin 1 - Address Bit 0 (A0)
Pin 2 - Address Bit 1 (A1)
Pin 3 - Address Bit 2 (A2)
Pin 4 - Address Bit 3 (A3)
Pin 5 - Address Bit 4 (A4)
Pin 6 - Address Bit 5 (A5)
Pin 7 - Address Bit 6 (A6)
Pin 8 - Address Bit 7 (A7)
Pin 9 - Address Bit 8 (A8)
Pin 10 - Address Bit 9 (A9)
Pin 11 - Address Bit 10 (A10)
Pin 12 - Address Bit 11 (A11)
Pin 13 - Address Bit 12 (A12)
Pin 14 - Address Bit 13 (A13)
Pin 15 - NC
Pin 16 - NC

SUMMARY TABLES & NOTES ON CONSTRUCTION:

Two tables are included with this article to summarize some of the information required to build this design. Table I is the "Package Summary List" identifying each IC and I/O socket position with its characteristics, use and power connections. Table II is a summary of the two interface sockets - position 1 (I/O#1) and position 2 (I/O#2). The information in table II identifies the signals associated with each pin of the interface plugs. The text below contains some comments on construction.

The prototype of this module was first built using the wrapped wire method of solderless interconnection. This method is described in complete detail in M.P. Publishing Co. publications #73-1 and #74-5. At the present time, October 12, 1974, there are no plans for making a PC card available for this module. A more advanced memory design which is in the process of construction at this time is expected to be available in PC form at a later date.

The process of wiring this module is straightforward and should present no major problems to those individuals using wire wrap techniques. The board requires 16 sockets in all, which can be neatly arranged in a 4x4 pattern of socket positions. The 8 memory chips can be grouped together in one half of the board. The only long wiring chains to be created are those which carry the address lines from socket position 2 to the address pins of the 8 memory circuits. It is probably best to wire these address busses first, then turn to the wiring of the less structured "random" logic of the page selection/control logic.

After the main wiring task has been accomplished for the RAM board, additional wiring will have to be supplied to connect the RAM to the CPU of the system. In a typical physical configuration of a card cage with back plane, this will involve adding connections in the back plane for the DIP jumper cables which go to the two I/O sockets. Two back plane socket positions are implied by this design - one for the address bus, and a second for the data bus plus miscellaneous power, ground connections and signal connections.

The wiring of the address jumper plug (socket position 8) is one of the last tasks to be performed before testing. This plug is a small DIP header plug with 8 pins in two rows of 4 each. Pins 7 and 8 have been wired to logic 1 and logic 0 respectively when you put the system together according to the logic diagram. This pair of pins supplies the "1" or "0" required for each of the page select inputs on pins 1 to 6. There are 64 possible page locations for the RAM module, with six bit binary tags from 000000 to 111111. It is suggested that if this is the first 256 byte page to be constructed it should be placed at binary page address 000001 - if it is the second, it should be placed at address 000010, etc. This way the memory address space of the 8008 CPU will be filled with active memory from the bottom up without holes.

In testing out this unit, begin by checking out all power voltages before plugging in the appropriate IC's. Then proceed to check out your wiring to make sure that all signals are reaching the module from the CPU. Finally, verify the operation of the memory by experimenting with the program suggested in the programming notes which follow.

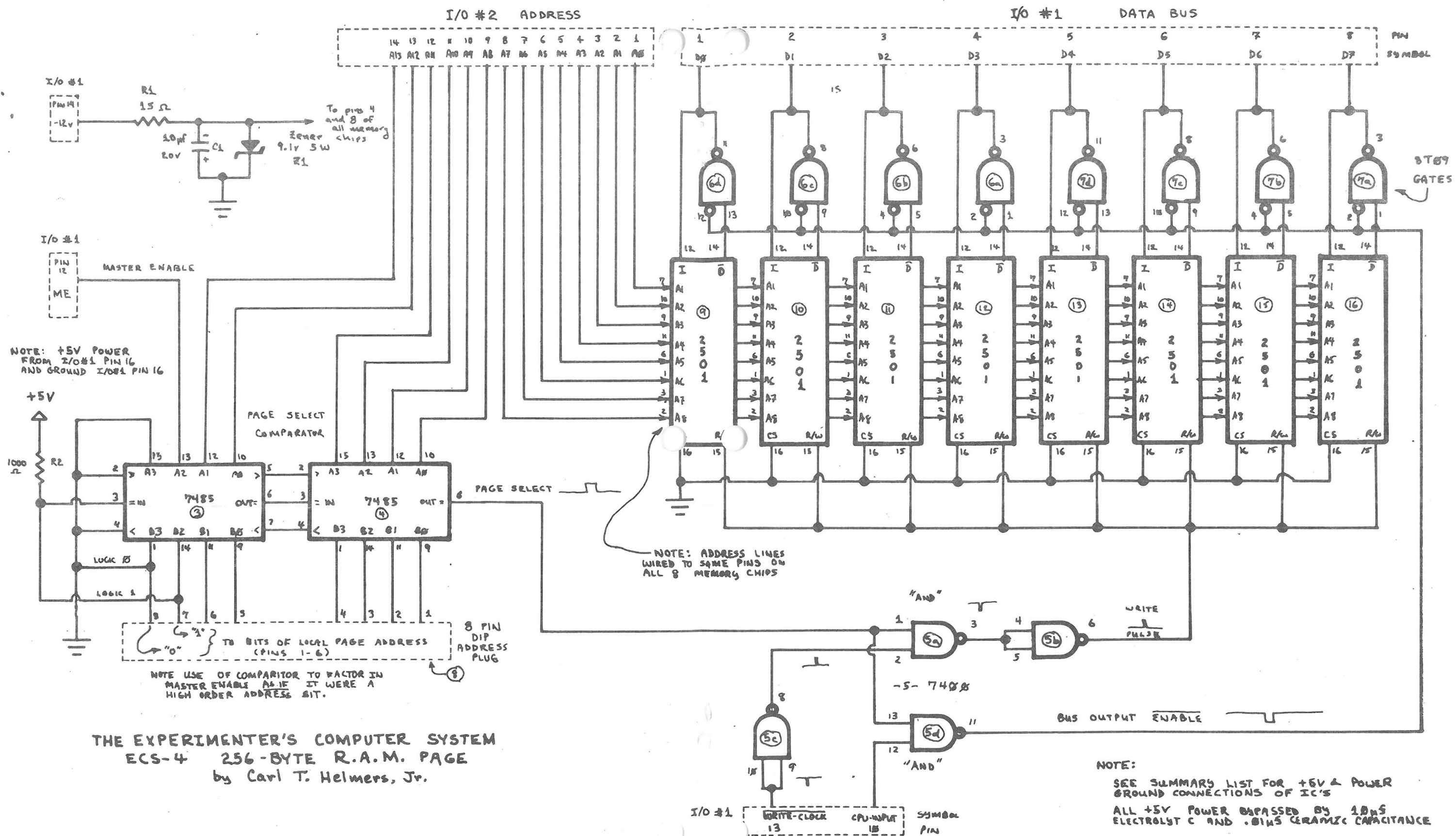


Figure #1

PROGRAMMING NOTES - TESTING THE MODULE:

The best way to test out a memory module given the fact that a CPU has been constructed is to use the CPU as a tool for examining the newly constructed module. The purpose of these notes is to discuss the use of the ECS-3 CPU design - the 8008 architecture plus the previously constructed 256-byte RAM Bootstrap memory - to check out additional memory. Since the ECS system as constructed up to this point still has not integrated peripherals, all system checkout must be done using the bare CPU and its indicator lamps - with programming done in absolute binary notation without any automated programming aids.

The problem is thus: verify that all 256 locations of the new page "work". To fit the definition of a working memory location, a given address must satisfy the following criteria:

1. It must be possible for the CPU to write data into the location at some point in time.
2. It must be possible for the CPU to read the data - without errors - at some later point in time.
3. The time interval between steps 1 and 2 should be of arbitrary length subject to the constraint that system power is not turned off in the interim.

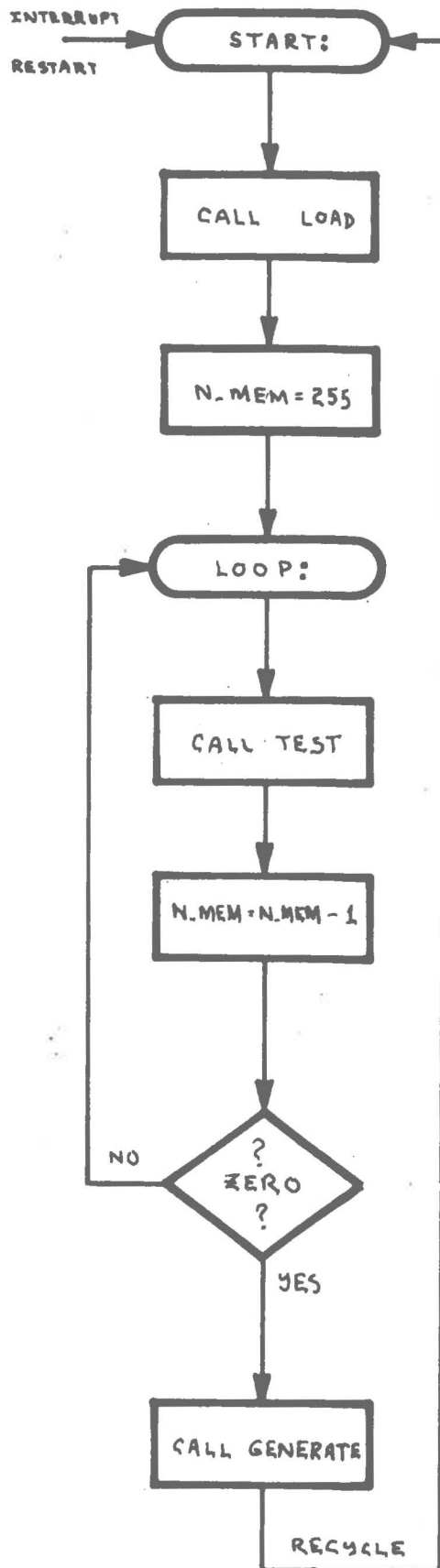
One way to accomplish a test of every word in the memory of the new module is to use a program whose broad outlines could be specified by the following verbal commands:

```
do forever;
  write a test pattern into the module;
  do for i = 1 to n (n arbitrary);
    check the test pattern & count errors;
  end;
  generate a new test pattern;
end;
```

(Here a notation for programming has been introduced which is similar to the computer language "PL/1" in which a loop is indicated by the word "do" and extends thru a matching "end". Indentation is used to show the "nesting" and keep track of which "end" matches which "do".)

The idea of the program is to repeatedly write test patterns into the memory then check them out "n" times to make sure that no bits are lost. The problem is thus completely specified in its general outlines - an abstract program which could theoretically be run on any computer with a memory module to be tested and not necessarily on the ECS-3 design's 8008 CPU. The problem is not complete - what remains to be done is to translate the abstract conceptual program into a specific set of binary instructions for the 8008 CPU to execute. The process of translation for this simple program will be performed by hand just as you will have to do with any programming application of the ECS series design if you do not have enough memory to run an assembler or interpretive computer language.

THE MEMORY TEST PROGRAM - DETAILS:



The code of the memory test program is found on pages 8 to 10 of this article. A flow chart is shown at the left on this page, providing a "roadmap" of program execution. Further comments are provided in the text on this page...

STARTING:

The memory test program begins when the Interrupt Pushbutton (see article ECS-3) is depressed - causing an "RST 0" instruction to be executed. The computer begins execution at location 0 following this restart. (Location 0 is labelled "START" in the program listing on page 8 of this article.)

LOADING MEMORY:

The first function performed by the program is to call the subroutine "LOAD" located at addresses 0020 to 002E (see page 9.) This routine places the 8-bit PATTERN (location 0061) into every word of the page being tested. (Locations 0048 and 0028 are flagged with asterisks to indicate that they are subject to change in setting up the program - they specify the page address of the page being tested and must not be set to 0.)

TEST LOOP:

The test loop consists of executing a memory test (CALL TEST) 255 times. The TEST routine scans every location of the desired page for agreement with PATTERN and adds 1 to ECOUNT (32 bits in locations 0063 to 0066) for each discrepancy. The subroutine ERROR does the multiple-precision arithmetic required.

GENERATING NEW PATTERN & RECYCLING:

Following the test loop in the main routine, a new pattern is created by adding INCREMENT to the old pattern in the routine called GENERATE. Then (as is the usual case,) the program branches back to START. This enables continued testing over night - or for a week if you want - integrating the total number of faults found over unlimited times. Whenever it is desired to check results of such long period testing, simply place the CPU in "single step" mode and put the bootstrap memory into "bootstrap mode" then look at the contents of locations 63 to 66. To continue operation, turn off "bootstrap mode" and place the CPU back in its "Run" mode.

MEMORY TEST PROGRAM - MAIN ROUTINE...

Addr	Type	Code	Description	
0000	I	106	START: CAL LOAD	} load a pattern in memory
0001	D	20	L(LOAD)	
0002	D	00	H(LOAD)	
0003	I	066	LLI	} N_MEM = 255
0004	D	60	L(N_MEM)	
0005	I	056	LHI	
0006	D	00	H(N_MEM)	
0007	I	076	LMI	
0008	D	FF	255	
0009	I	106	LOOP: CAL TEST	} count errors in pattern
000A	D	40	L(TEST)	
000B	D	00	H(TEST)	
000C	I	066	LLI	} N_MEM = N_MEM - 1
000D	D	60	L(N_MEM)	
000E	I	056	LHI	
000F	D	00	H(N_MEM)	
0010	I	317	LBM	
0011	I	011	DCB	
0012	I	371	LMB	
0013	I	110	JFZ LOOP	} repeat test until N_MEM=0
0014	D	09	L(LOOP)	
0015	D	00	H(LOOP)	
0016	I	106	CAL GENERATE	} create a new test pattern
0017	D	30	L(GENERATE)	
0018	D	00	H(GENERATE)	
0019	I	104	JMP START	} keep cycling indefinitely
001A	D	00	L(START)	
001B	D	00	H(START)	

The following are alternate definitions for the end of program in locations 0019+...

0019	I	000	HLT	- halt after one test cycle
0019	I	005	RST 0	- using RST as a JMP 0

MEMORY TEST PROGRAM - GENERATE ROUTINE...

0030	I	056	GENERATE: LHI	} PATTERN=PATTERN+INCREMENT
0031	D	00	H(PATTERN)	
0032	I	066	LLI	
0033	D	61	L(PATTERN)	
0034	I	307	LAM	
0035	I	060	INL	
0036	I	207	ADM	
0037	I	061	DCL	
0038	I	370	LMA	
0039	I	007	RET	

MEMORY TEST PROGRAM - LOAD ROUTINE...

Addr	Type	Code	Description
0020	I	066	LOAD: LLI
0021	D	61	L(PATTERN)
0022	I	056	LHI
0023	D	00	H(PATTERN)
0024	I	307	LAM
} reg-a = PATTERN			
0025	I	066	LLI
0026	D	00	L(MEMPAGE)
0027	I	056	LHI
*0028	D	01	H(MEMPAGE)
} set up memory page start address for test			
0029	I	370	LOAD_LOOP: LMA
002A	I	060	INL
002B	I	110	JFZ LOAD_LOOP
002C	D	29	L(LOAD_LOOP)
002D	D	00	H(LOAD_LOOP)
002E	I	007	RET
} write memory word ← calculate next address } scan entire page			

MEMORY TEST PROGRAM - TEST ROUTINE...

0040	I	066	TEST: LLI
0041	D	61	L(PATTERN)
0042	I	056	LHI
0043	D	00	H(PATTERN)
0044	I	307	LAM
} reg-a = PATTERN			
0045	I	066	LLI
0046	D	00	L(MEMPAGE)
0047	I	056	LHI
*0048	D	01	H(MEMPAGE)
} set up memory page start address for test			
0049	I	277	TEST_LOOP: CPM
004A	I	112	CFZ ERROR
004B	D	80	L(ERROR)
004C	D	00	H(ERROR)
} IF MEMPAGE _i NOT = PATTERN THEN CALL ERROR			
004D	I	060	INL
004E	I	110	JFZ TEST_LOOP
004F	D	49	L(TEST_LOOP)
0050	D	00	H(TEST_LOOP)
0051	I	007	RET
} Scan entire page			

MEMORY TEST PROGRAM - DATA AREAS...

N_MEM:	0060	- This location is an 8-bit variable which holds the index for the main program's test loop.
PATTERN:	0061	- This location is an 8-bit variable which is initialized to a starting test pattern and is modified by the generate routine.
INCREMENT:	0062	- This location is an 8-bit variable which is initialized to an odd integer value.
ECOUNT:	0063 - 0066	- 32 bit error count, initialized to zeros.

MEMORY TEST PROGRAM - ERROR ROUTINE...

Addr	Type	Code	Description
0080	I	066	ERROR: LLI
0081	D	66	L(ECOUNT+3)
0082	I	056	LHI
0083	D	00	H(ECOUNT)
0084	I	006	LAI
0085	D	01	1
0086	I	207	ADM
0087	I	370	LMA
} add 1 to the least significant byte of the 32 bit error count and set carry for propagation to next byte's add.			
0088	I	066	LLI
0089	D	65	L(ECOUNT+2)
008A	I	006	LAI
008B	D	00	0
008C	I	217	ACM
} add zero to second byte with carry input from first add.			
008D	I	066	LLI
008E	D	64	L(ECOUNT+1)
008F	I	006	LAI
0090	D	00	0
0091	I	217	ACM
} add zero to third byte with carry input from second add.			
0092	I	066	LLI
0093	D	63	L(ECOUNT+0)
0094	I	006	LAI
0095	D	00	0
0096	I	217	ACM
} add zero to fourth byte with carry input from third add.			
0097	I	003	RFC
← return if less than 2^{32} errors...			
0098	I	000	HLT
← halt if more than $2^{32}-1$ errors...			

NOTES ON FUTURE ISSUES...

At the time this article is going to press, the prototype of the fifth article's hardware design is being assembled. ECS-5, entitled "I/O Controller", contains the details of a general I/O decoding scheme for the Experimenter's Computer System. This scheme involves responding to the 32 possible output ports and 8 possible input ports of the 8008 - with provision of an 8-level interrupt system accessed via input port 0. To provide an example of input and output hardware, the design originally to be described in ECS-7 has been moved ahead and included with ECS-5. In its place, article ECS-7 will be devoted exclusively to the keyboard driven memory editor software topic. Also included with the ECS-5 article is the description of a very simple LED output display register.

The memory editor program being designed for ECS-7 will be a fairly general tool useful in your programming of the Experimenter's Computer System.

The Experimenter's Computer System: Part 5

I/O CONTROLLER
AND SIMPLE I/O DEVICE PROTOTYPES

by Carl T. Helmers, Jr.

INTRODUCTION:

This article is the fifth number in the Experimenter's Computer System series. It continues the description of hardware begun in the earlier articles by providing information on the following subsystems and their use:

I/O Instruction Decode - logic to detect CPU I/O instruction states and create bus enable (input) or data transfer clock (output) information for all I/O ports.

Interrupt Management Logic - an 8-bit interrupt flag register with associated interrupt control operations is used by programs to determine interrupt sources and to mask interrupts during critical operations.

ASCII Keyboard Input Device - one interrupt device and its associated interrupt are shown connected to an ASCII keyboard input encoded via a diode matrix.

Binary Display Devices - two output ports (without interrupts) are implemented as binary display registers - a total of 16 LED's which can be controlled by a program.

Simple Interrupt Handler - the article provides the listing of a simple interrupt handler program used to decode keyboard interrupts, read the keyboard and display the bit pattern read in the two display output ports.

Binary Calculator Program - a simple calculator program which will add and subtract 16 bit numbers entered from the keyboard and display results in the LED output ports.

The information contained in this article assumes a familiarity with the Experimenter's Computer System concepts and terminology, particularly the information contained in article ECS-3 previously published.

CHANGES IN DESIGN CONCEPT:

The original intention (see ECS-1) was to use an address decoding method of I/O for the Experimenter's Computer System, in which I/O is done by memory reference to a selected page of memory as in several large scale computing systems. The concept is a beautiful one - but unfortunately its implementation on the 8008 CPU based Experimenter's Computer System is inappropriate. Thus the present article describes a generalized I/O controller which makes use of the 8008's I/O instruction format and provides the control signals needed to manage all 32 output ports and 8 input ports.

INSTRUCTION DECODE AND CLOCKING:

The Instruction Decode and Clocking logic is shown in drawing #1 of this article. When the CPU executes an I/O instruction, the Address Latch always receives the information needed to decode a PCC cycle and the I/O unit involved, as well as the old contents of the CPU's accumulator (register "A"). This logic reacts to the A9 through A15 bit pattern in order to detect an I/O PCC cycle and enable clock or bus control information to pass through the decoding network to the individual device selected by the I/O instruction.

OPERATION DECODE is provided by the 7442 selector labeled -6- in the drawing. This device accepts a 4-bit pattern from A15 through A12, of which the states labeled 4, 5, 6 and 7 are significant for I/O. (Internal logic designations of the 7442 - and other IC's - are on the inside of its symbol, with external pins on the outside of the symbol.) The output pins for these states (4, 5, 6 & 7) represent detection of an I/O cycle for ports 0, 1, 2 & 3 respectively.

INPUT PORT BUS ENABLE is provided by the 7442 labeled -11-. If an input operation is indicated, the state 4 output of pin -6.5- will produce a "1" input to -12.13- which is anded with the CPU-INPUT signal of -12.12- producing a low input to -11.12-. The "D" input -11.12- of the bus enable selector serves as a gate for all outputs - and if 0 enables one of the 8 input ports selected by address lines A9 to A11. The output of -11- can be used to directly control bus interface gates of input units - as for instance is shown by -11.1- which is connected to the Interrupt Port (INO) and -11.2- which is connected to the Keyboard Port (IN1). The remaining outputs for other devices are brought to an I/O connector for wiring to additional devices through the backplane of the CPU.

OUTPUT PORT DATA TRANSFER CLOCK ROUTING is provided by the set of four 7442 selectors labeled -7-, -8-, -9- and -10-. Output is accomplished by writing the content of the A0 through A7 address latch lines into a device register with the CT3 pulse of an I/O cycle. The 32 output lines of these four 7442's route the CT3 pulse of I/O to the appropriate devices - locally for the interrupt control and display register outputs, and via connectors and the CPU backplane for devices which will be added later.

For each output port, a NAND gate is used to "AND" together the CT3 pulse and the appropriate port selection of operation decode IC -6-. The result is a "0" enabling pulse on the appropriate device selection 7442's D line - resulting in a negative logic pulse on the device addressed by A9 to A11.

INTERRUPT MANAGEMENT LOGIC:

Drawing #3 contains the logic of the Interrupt Flag Register and its associated control logic and bus interface. As wired in ECS-3, the CPU has only one possible interrupt and only one possible hardware interrupt operation - a restart at location 0 of the memory. (This is not the only scheme possible, but has attractions in that only one possible program can be directly invoked by hardware - the instructions starting at location 0 and its logical successors. The disadvantage of this scheme is that there is a time and memory penalty to be paid in the software interrupt decode which will be used.) This original interrupt scheme is retained and augmented by the interrupt management logic of the I/O Controller.

In order to find out what device has asynchronously (with respect to program operation) called the CPU for some interrupt action, a set of interrupt flags - the "Interrupt Flag Register" - is provided. These flags are implemented as sections of 7473 dual flip-flops. The interrupt pulse of the device in question sets the corresponding flag "on". At any time, under program control, the Interrupt Flag Register can be read to find out the status of pending interrupts, using the INO (input port zero) instruction. In the present article, only the keyboard device has an interrupt connected - and as a result, all the unused interrupt inputs (I/O socket pins 1.10 to 1.16) must be grounded to avoid setting the associated flags with transient noise. When the Interrupt Flag Register has any flag on, at least one of the inputs to the 7430 -31- will be zero. Since the negative logic "or" function is provided by this 8-input NAND, its output is logical "1" if any interrupt is pending.

If the Interrupt Enable Flip Flop formed by section -13b- and -30c- of NAND integrated circuits is logical "1", the NAND gate section -13c- enables the fact that an interrupt is pending to reach the CPU via I/O socket pins 1.1 and 1.2, which must be connected via the backplane to the CPU's interrupt jumper plug, drawing 8 of article ECS-3. The interrupt service routine at location 0 of the computer's memory must do the following:

1. Output a command to disable further interrupts, simultaneous with the input of the current interrupt flag register content. (INO command preceded by loading accumulator with "10" in bits 1 and 0 respectively.)
2. Use the interrupt flag register information just read to decode the pending device or devices - the interrupt service routine proper.
3. Output a command to enable further interrupts, followed by a return to the interrupted program. (INO instruction preceded by loading accumulator with "11" in bits 1 and 0 respectively, followed by a RET instruction.)

INTERRUPT CONTROL OPERATIONS:

The status of the interrupt operation is controlled by the Interrupt Enable Flipflop in drawing #3 as mentioned above. Output port 0 (which corresponds to input port 0) is used to set and reset this flag bit under program control. If the "INO" instruction is executed with the bit pattern "11" in the accumulator's low order bits, then output port 0

logic of drawing #3 will present a pulse on the "I-ENABLE" line to set the control flipflop. If the "INO" instruction is executed with the bit pattern "10" in the accumulator's low order bits, then output port 0 logic will present a pulse on the "I-DISABLE" line to reset the flipflop and cut-off further CPU interrupts until it is set.

Since this interrupt operation requires intimate program involvement for its control, a third input on the "set" side of the control flip flop is provided. This third input is a time delay network which accomplishes two ends:

1. It allows manual intervention to enable interrupts in the event of a software bug which "locks up" the system.
2. By virtue of the fact that the time delay does not allow a logic "1" input to the Interrupt Enable Flip Flop's manual reset until several milliseconds after power turn on (with all other inputs defined "instantaneously"), this guarantees that when turning the system on interrupts will be enabled initially.

Switch S1 can be a pushbutton switch on your CPU panel - and can be the same switch which was formerly used for the manual interrupt to the CPU when you built and tested ECS-3 without I/O devices. Since the interrupt jumper used previously no longer is connected to the manual interrupt Switch in ECS-3, this substitution is possible.

Note that the output port 0 (INO) logic of this diagram treats the content of bits 2 through 7 as "don't care" states. Further, since only the bit patterns "11" and "10" in bits 1 and 0 are recognized as described above, it is possible to output arbitrary information in bits 2 through 7 so long as bits 0 and 1 are left "0" - without affecting the interrupt status.

Finally, note that the "I-ENABLE" pulse used to set the Interrupt Enable Flip Flop after an interrupt is serviced will also clear the Interrupt Flag Register. This sets up the register for future interrupt events - and makes the additional requirement on the interrupt service routine that it service all pending interrupts. In servicing pending interrupts, the concept of "priority" is a useful one. "Priority" is the order in which simultaneously read interrupts are serviced - a "high priority" device should be serviced ahead of a "low priority" one. Priority with this scheme of operation is determined by the order in which the interrupt service routine checks the individual bits it has read from the interrupt flag register. A high priority (ie: time critical, such as the tape recorder device) I/O operation is checked and serviced first, while a lower priority device such as the keyboard can wait for ages - many milliseconds - without losing information, so it is the last (or nearly the last) interrupting device to be checked by the interrupt service routine when multiple devices make use of this facility. The programs which are included with this article have only one interrupt possible, the keyboard interrupt.

BINARY DISPLAY OUTPUT DEVICE:

Drawing #2 shows the logic needed to implement the two simple 8-bit LED display devices tied to output ports 30 and 31. The design shown uses 74100 devices to store data during an output operation and to directly drive the LED displays. An alternative is to use 7475 devices (four would be required instead of two 74100's) - with appropriate changes in wiring and sockets.

When the appropriate OUT30 or OUT31 instruction is executed by a program running in the CPU, the clock line input to the register receives a pulse at CT3 time, transferring the content of A0 to A7 into the "device register" of the 74100 - and displaying the result on the LED's. Note that the sense of the outputs will be inverted when data is written in this hardware - a "1" will turn off the light, and a "0" will turn on the light in a given bit position. There are several ways to achieve a pattern with "on" indicating a "1" bit...

- when you wire your controller, add inverters on the inputs to the output device registers as indicated in drawing #2 by the dotted line and its note.

- use a set of 7475 devices instead of two 74100's and drive the lamps with the complement outputs of the 7475.

- or - as is always the case - you can correct for hardware foibles with programming. When ready to display, invert the data in the accumulator first, for example:

LAM	get the data into A from memory
XRI	exclusive or with all "1's"
"FF"	inverts the data
OUT30	write the data to the device.

The diagrams indicate the LED's as a binary display device. There are other display format choices which may be used instead of the simple binary lamps. For instance, if you use MAN-4 hexadecimal readouts with an appropriate decoder network hex digits can be displayed. As another example, if groups of three bits are fed through 7447 decoders, octal display is possible. Similarly, there is no need to use a lamp display at all if you want to do something else - for instance another copy of this I/O device could be used to control 8 relays through appropriate drivers, or as inputs to an 8-bit digital to analog conversion in situations where a voltage level output is desired.

ASCII KEYBOARD INPUT DEVICE:

Drawings #2, #4 and #5 detail a simple "diode matrix" keyboard device and its input to the computer via device "1" of the input port. The actual matrix is shown with a full set of 63 codes and corresponding keyswitches - however in actuality, only those keys which have a teletypewriter keyboard position will be wired. In addition to the 6-bit ASCII used for data keys, the "CTRL" and "SHIFT" keys are used to provide the two additional bit lines required for a full 8-bit input.

The output of the keyboard matrix is in positive logic ASCII form, but with a default (no key pressed) state of "1" on all input lines to the keyboard logic of drawing #2. Thus direct input of ASCII code 63 is not possible - since detection of keystrokes is accomplished by finding changes from the all "1"s state. The "1" state is present when no key is depressed due to the "pulling up" action of resistors R29 to R34. When any key is depressed, the bit pattern is selected by the diodes in the matrix pulling down the normal level to approximately 0 volts, through the diode junctions. Contact bounce would be expected to cause problems through the receiving 7404 inverters on drawing #2 - however an RC time constant provided by a 100K resistors and .1 mfd condenser in each case smooths the key bounce associated with a keystroke. The output of the inverters is a "negative logic" ASCII code which is normally all "0"'s and has at least one non-zero bit if codes 0 through 62 are selected. The 6 bits are "orred" together with the combination of NOR and NAND functions shown in drawing #2, producing a "key on" signal. This signal is delayed by the RC network between inverters -16f- and -16e- to produce an interrupt pulse out of the oneshot -21- several 10's of milliseconds after the key was depressed. This assures that stable data will be present at the bus interface when the CPU responds to the interrupt, since it allows for "slop" in the debounce networks due to component tolerances. The CPU must respond to the interrupt by performing an "IN1" operation to read the key. Since the one-shot which generates the interrupt will only respond to the single rising edge of the "key on" output of -16e-,, only one interrupt will be generated - even if two or more keys are depressed in an overlapping fashion.

For this simple input keyboard design, an archaic kluge is used for the encoding function - the diode matrix. This is one way of getting keyboard input, one of the first ever used in computers. It has several disadvantages which might be noted:

- Multiple keys can be depressed simultaneously, with the result being a logical "or" of their bit patterns.
- There is no rollover feature to interpret multiple overlapping keystrokes.
- It requires a large number of components - 192 diodes for a full matrix as shown.

In spite of the technical disadvantages, this form of a keyboard is described at the present time due to the fact that it requires no special LSI components to build and can be wired directly with surplus diodes.

A better technical solution of the keyboard input problem is to use one of several forms of keyboard encoding chips available on the new equipment markets, or the surplus assemblies often sold. If you buy a surplus keyboard there are two possibilities: if you are lucky, the keyboard will work as is and can be figured out. If you are unlucky, you will have purchased an array of key switches - and will have to build the diode encoder or its equivalent anyway. Assuming you have such a keyboard, the interface is simple - it will have parallel bit lines output. These are routed to the bus interface 8T09's. The interrupt oneshot should be triggered off a "key on" output of the board, or logic similar to drawing #2's key-on logic can be used instead.

Note that most keyboards do not have a full set of 63 ASCII codes possible - a typical typewriter has only 44 to 48 separate keys. The diagrams have noted along the bottom the lower case characters of a typical Teletype keyboard arrangement - with a "check" indicating that the character in question is present on the TTY type keyboard layout, as a separate key switch. To generate true ASCII from the keyboard, some software modification will be required for upper case and lower case characters, as well as "CTRL" characters. Table I shows the correspondence between input codes from the keyboard shown and data in a true 6-bit ASCII format, for all 63 of the possible codes in the diode matrix.

Also, note that there is no "carriage return" code indicated in drawings #4 and #5. The same applies as well for "line feed" and "escape". This leads to the general topic of special purpose keyboards and additional keys. Basically, any one of the unused codes of a Teletype style keyboard can be used for the implementation of additional input key possibilities. In table I, the "CR", "LF" and "ESC" keys are also indicated with no 6-bit ASCII correspondences. Any number of additional key arrays can be wired into the matrix either in parallel with keyboard switches or to the unused codes of the matrix's TTY format ninputs. This feature will be used in future articles to implement special purpose keyboards for use in calculator applications and in computer game applications.

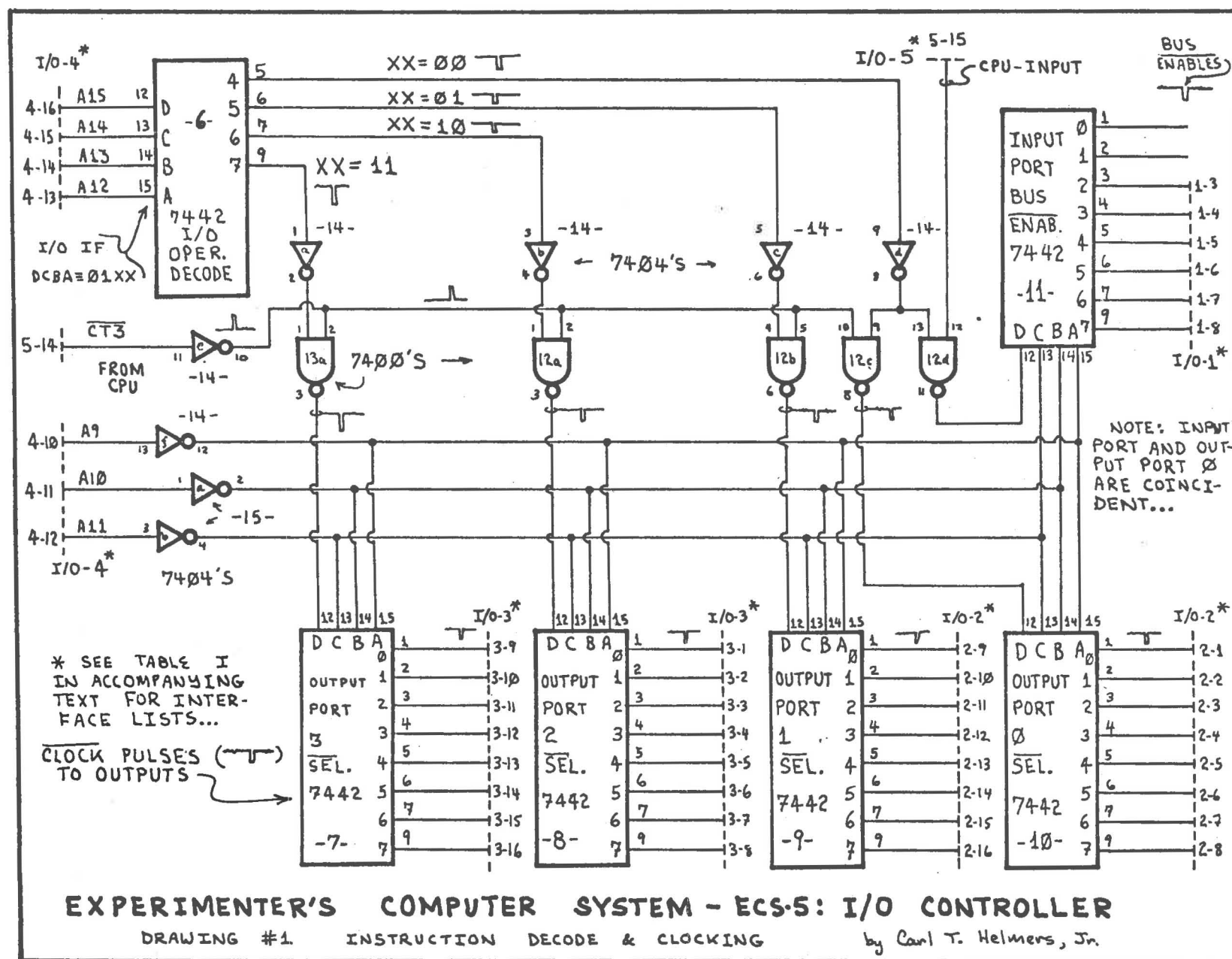
CONSTRUCTION:

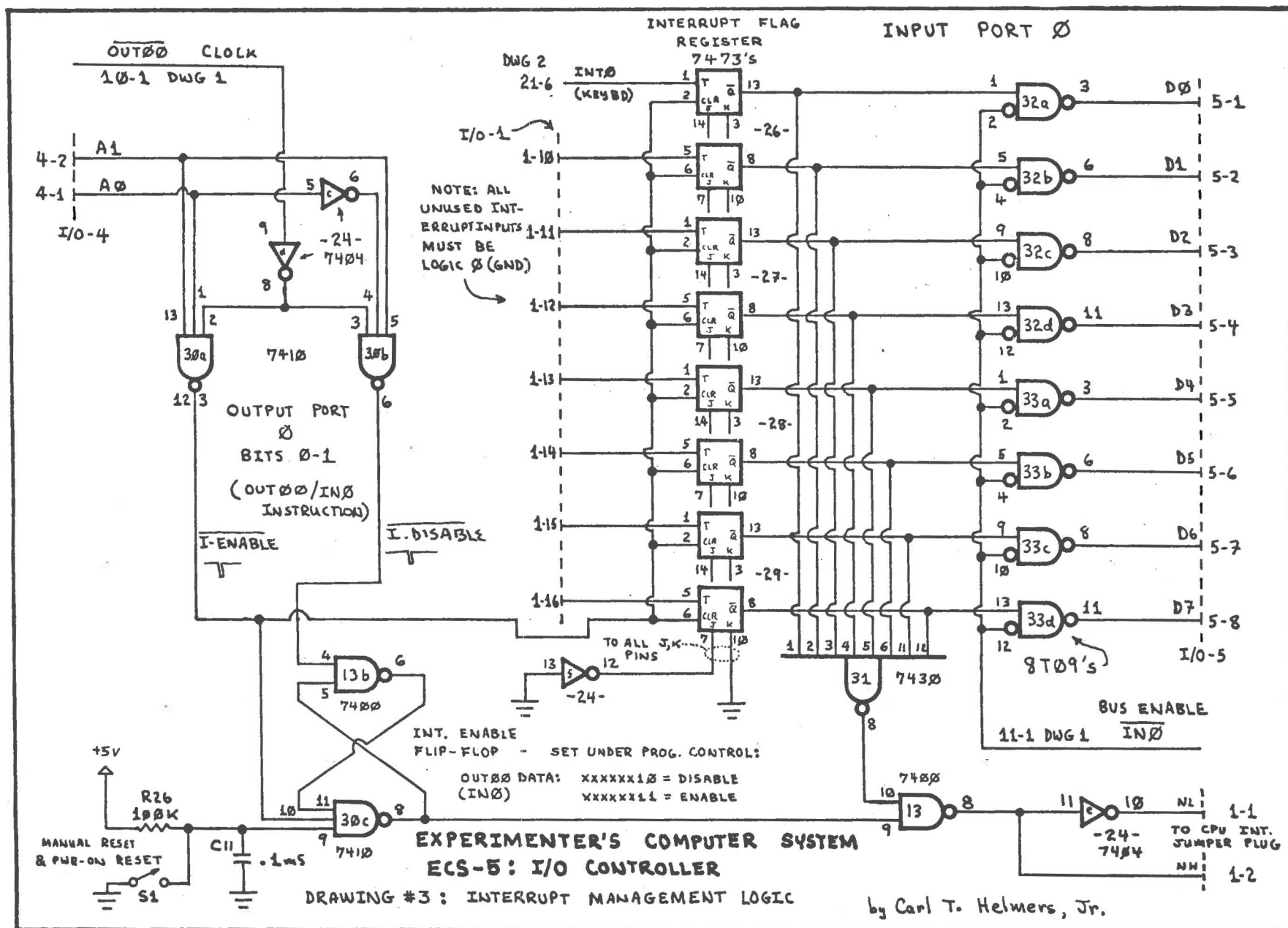
The best way to put together the I/O controller is to use the wire wrap technique of construction as described in publication #73-1 and its supplement #74-5. At the present time (November 16 1974) there are no plans to make this particular design available in PC form, since it is a "one of a kind" item in any implementation of this computer.

The article contains five complete drawings of the logic for this design on the pages which follow, supplemented by the following tables:

- Table I: Character codes
- Table II: IC Package & I/O Socket Summary List
- Table III: Parts List
- Table IV: I/O Pin Assignment Lists

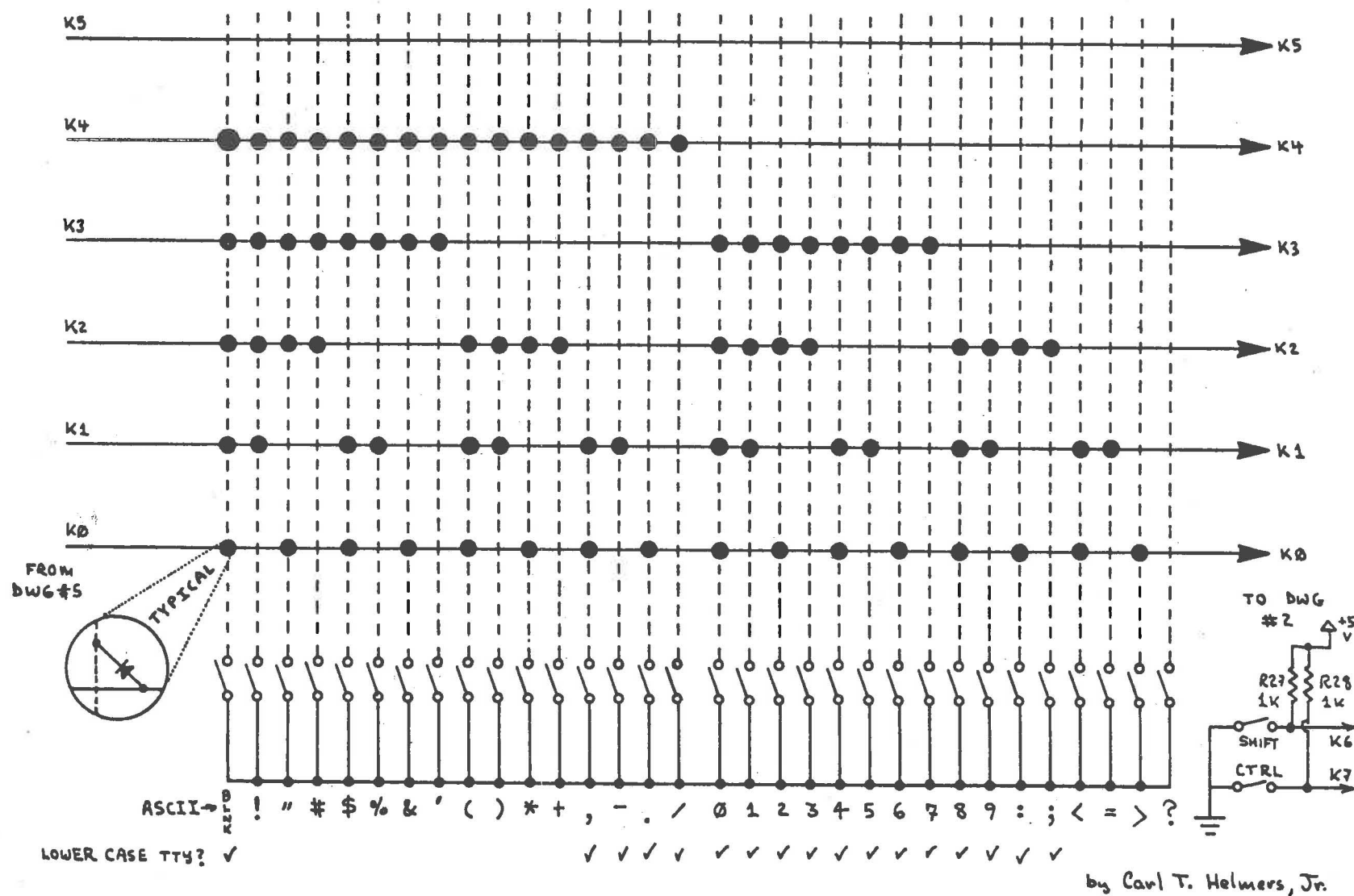
The testing of this circuit can be accomplished using the simple interrupt service program " **KEYBOARD-ECHO**" described following the tables and drawings. Then, once you have the basic circuit in operation, you may wish to try the simple "BINARY-CALCULATOR" program shown at the end of the article - a program which will enable you to enter binary digits, add your entries to an accumulator of 16 bits (displaying the result), subtract entries, clear entries or clear the accumulator.



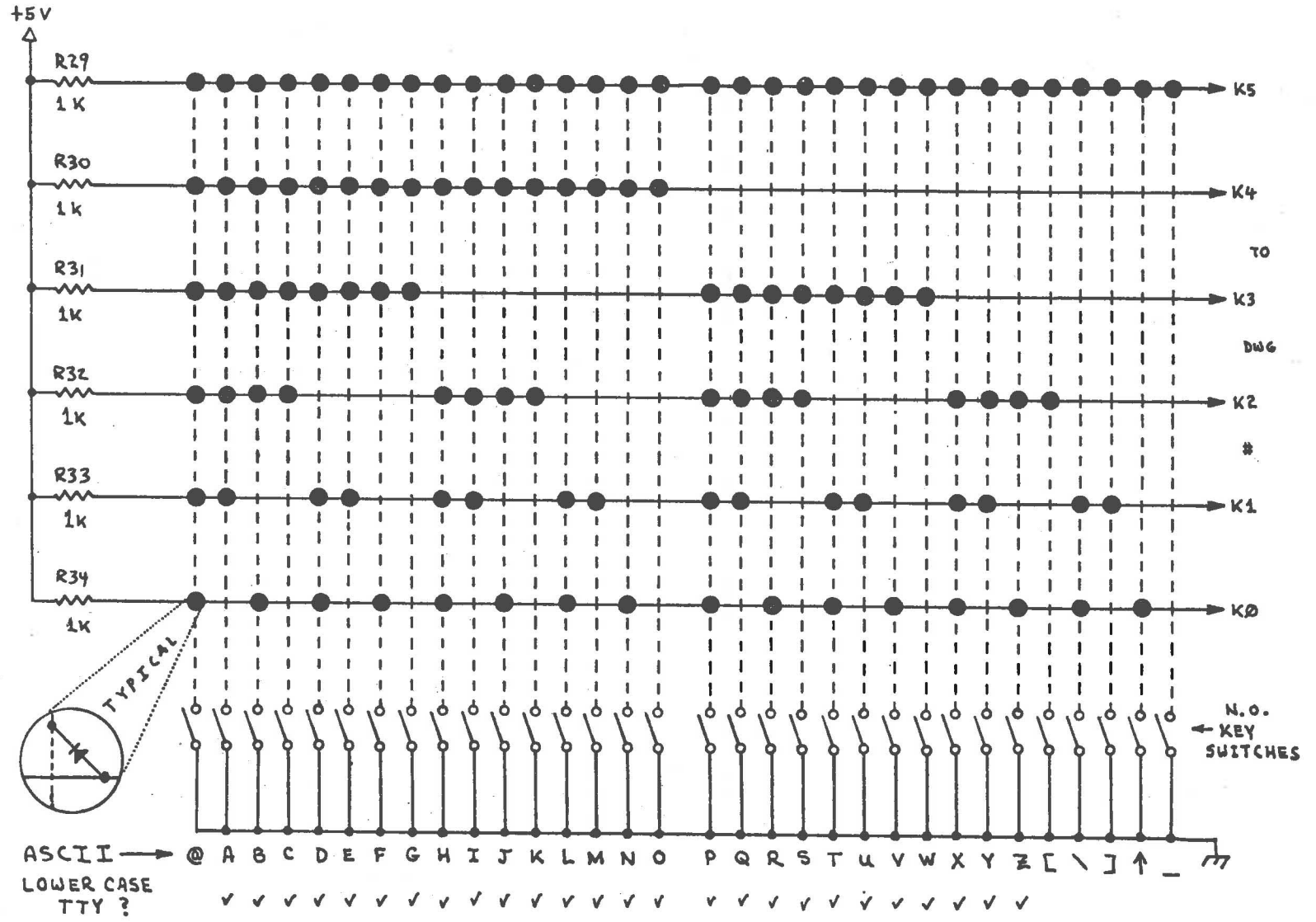


EXPERIMENTER'S COMPUTER SYSTEM — ECS-5: I/O CONTROLLER

DRAWING #4: ASCII KEYBOARD ENCODER MATRIX - PART 1



EXPERIMENTER'S COMPUTER SYSTEM: ECS-5 I/O CONTROLLER DRAWING #5 - ASCII KEYBOARD ENCODER MATRIX - PART 2



by Carl T. Helmers, Jr.

T A B L E I :
Keyboard Character Codes

ASCII Symbol	Binary Input	Hex Code	Symbol	Binary Input	Hex Code
@	11000000*	C0	blnk	11100000	E0
A	11000001	C1	!	11100001	E1
B	11000010	C2	"	11100010	E2
C	11000011	C3	#	11100011	E3
D	11000100	C4	\$	11100100	E4
E	11000101	C5	%	11100101	E5
F	11000110	C6	&	11100110	E6
G	11000111	C7	'	11100111	E7
H	11001000	C8	(	11101000	E8
I	11001001	C9	)	11101001	E9
J	11001010	CA	*	11101010	EA
K	11001011	CB	+	11101011	EB
L	11001100	CC	,	11101100	EC
M	11001101	CD	-	11101101	ED
N	11001110	CE	.	11101110	EE
O	11001111	CF	/	11101111	EF
P	11010000	D0	ø	11110000	F0
Q	11010001	D1	1	11110001	F1
R	11010010	D2	2	11110010	F2
S	11010011	D3	3	11110011	F3
T	11010100	D4	4	11110100	F4
U	11010101	D5	5	11110101	F5
V	11010110	D6	6	11110110	F6
W	11010111	D7	7	11110111	F7
X	11011000	D8	8	11111000	F8
Y	11011001	D9	9	11111001	F9
Z	11011010	DA	:	11111010	FA
[	11011011	DB	;	11111011	FB
\	11011100	DC	< (LF)	11111100	FC
]	11011101	DD	=	11111101	FD
↑	11011110	DE	> (CR)	11111110	FE
_	11011111	DF	?	11111111	FF**

Notes:

*1. With "shift" key and "ctrl" key not depressed, the codes will be as shown. Shift turns off bit 6 and ctrl turns off bit 7.

**2. The question mark code has no direct input since it is the null position for the keyboard's output.

3. In programming, true six-bit ASCII is obtained for all codes directly wired to keys by masking off bits 6 and 7 of the input with an "and" CPU instruction. For translation of shifted and control forms of characters, one method is to look up the desired 6-bit code in a table addressed by the actual low order pattern read. The shift and control keys must be held simultaneously with depression of the shifted character so that the code is present when read.

T A B L E I I :

IC Package & I/O Socket Summary List

#	Description of Socket Position		Pins	+5 V	Gnd.
1	I/O-1	Bus Enables & Interrupts	16	-	-
2	I/O-2	Ports 0 & 1 Write Clocks	16	-	-
3	I/O-3	Ports 2 & 3 Write Clocks	16	-	-
4	I/O-4	Address Inputs	16	-	-
5	I/O-5	Data Bus, Power, Misc. CPU	16	-	-
6	7442	Operation Decode	16	16	8
7	7442	Port 3 Clock Routing(Output)	16	16	8
8	7442	Port 2 Clock Routing(Output)	16	16	8
9	7442	Port 1 Clock Routing(Output)	16	16	8
10	7442	Port 0 Clock Routing(Output)	16	16	8
11	7442	Input Port Bus Enable Source	16	16	8
12	7400	Operation Decode Logic	14	14	7
13	7400	Misc. NAND Logic	14	14	7
14	7404	Operation Decode Logic Inverts.	14	14	7
15	7404	Op. Decode & Misc. Logic Inverts.	14	14	7
16	7404	Misc. Inverts (Kbd. Input)	14	14	7
17	7402	Key On Detection Logic	14	14	7
18	7420	Key On Detection Logic	14	14	7
19	8T09	Keyboard Bus Interface Gate 4-7	14	14	7
20	8T09	Keyboard Bus Interface Gate 0-3	14	14	7
21	74121	Keyboard Interrupt Oneshot	14	14	7
22	74100	Output Port 30 Latch (see text)	24	24	7
23	74100	Output Port 31 Latch (see text)	24	24	7
24	7404	Misc. Inverters	14	14	7
25	I/O-6	Keyboard Interface Plug	16	16	9
26	7473	Interrupt Flags 0-1	14	4	11
27	7473	Interrupt Flags 2-3	14	4	11
28	7473	Interrupt Flags 4-5	14	4	11
29	7473	Interrupt Flags 6-7	14	4	11
30	7410	Output Port 0 Logic	14	14	7
31	7430	Interrupt Detection Logic	14	14	7
32	8T09	Interrupt Flag Reg Bus Int. 0-3	14	14	7
33	8T09	Interrupt Flag Reg Bus Int. 4-7	14	14	7
34	-	Discretes on Carrier: R1 to R8	16	-	-
35	-	Discretes on Carrier: C1 to C8	16	-	-
36	-	Discretes on Carrier: R10 to R17	16	1 to 8	-
37	-	Discretes on Carrier: R18 to R25	16	1 to 8	-
38	-	Discretes on Carrier: L1 to L8	16	-	-
39	-	Discretes on Carrier: L9 to L16	16	-	-
40	-	Discretes on Carrier: Miscellany	16	-	-

Note: See table IV for list of I/O socket pins & use.

T A B L E I I I:

Parts List

R1 to R9 100K @ $\frac{1}{4}$ W
 R10 to R25 220 @ $\frac{1}{4}$ W
 R26 100K @ $\frac{1}{4}$ W
 R27 to R34 1000 @ $\frac{1}{4}$ W

C1 to C8, .1 mfd 50v
 C10 & C11 ceram. 20%
 C9 .001 mfd

L1 to L16 LED, 10ma @ 1.6v

S1 SPST Button(NO)

Miscellaneous:

Full Diode Matrix requires 192
 switching diodes (eg: 1n914) or
 any handy surplus diodes.)

Vector "P" pattern board can be
 used as basis for diode matrix
 (requires about 4" by 16")

Keyboard required is magnetic reed
 or equivalent, TTY layout.

40 Sockets are shown in table I.

In addition, you will need plugs and cables to carry I/O to the backplane, plus an appropriate addition to backplane wiring to receive these cables. In making the device, don't forget to make the LED display bvisible - one handy way of mounting is to make a smoked glass cover (with cutout) for the keyboard, with the LED's in a line visible behind it.

T A B L E I V :

I/O Pin Assignment Lists

I/O-1: Bus Enables & Interrupts

1. Interrupt! Norm. Low to CPU
2. Interrupt! Norm. Hi to CPU
3. to 8. - respectively Input
Ports 2 to 7 bus enables.
9. NC
10. to 16 - respectively Interrupt 1 to 7 sources (0 is key board.)

I/O-2: Ports 0 and 1 Clocks

1. to 8. - respectively Output
Ports 00 to 07 clocks.
9. to 16. - respectively, Output
Ports 10 to 17 clocks.

I/O-3: Ports 2 and 3 Clocks

1. to 8. - respectively, Output
Ports 20 to 27 clocks.
9. to 16. - respectively, Output
Ports 30 to 37 clocks.

I/O-4: Address Lines.

1. to 16. - Lines A0 to A15, respec.

I/O-5: Bus/Power/Misc.

1. to 8. - respectively, Data Bus
lines D0 to D7.
9. Ground - power
10. to 13. - no connection
14. CT3 SIGNAL from CPU*
15. CPU-INPUT from CPU
16. +5 volt power.

*Note: Thru oversight, this
 signal was not mentioned as
 output from CPU in ECS-3.
 Use an extra I/O-1 pin (see
 ECS-3, page 14) for this
 line.

I/O-6: Keyboard

1. to 8. - respectively K0 to K7.
9. Ground
10. to 15. No Connection
16. Power - +5 volts

KEYBOARD-ECHO Program: This program will respond to interrupts from the keyboard input device by reading the bit code presented (over and over again as long as the key first pressed or any other key is held down) and displaying the code in both halves of the 16 bit display device. Use the program when initially checking out the I/O hardware.

```

                                INTERRUPT:
0000  I   006      LAI           Set up disable code for
0001  D   02      '00000010'      output to interrupt logic.
0002  I  101      INO           Read IFR & disable interrupts
0003  I  012      RRC           Set Carry from A0 (INT0 flag)
0004  I  140      JTC           }
0005  D  20      L(KEYSERV)      } Got to Keyboard Routine if
0006  D  00      H(KEYSERV)      } keyboard interrupt.
0007  I  006      LAI           }
0008  D  00      00             } Turn on all LED's for error
0009  I  161      OUT30          } indication (see p. 5)
000A  I  163      OUT31          }
000B  I  006      LAI           }
000C  D  03      '00000011'      } Set up enable code for
000D  I  101      INO           } output to interrupt logic.
000E  I  377      HALT          } Clear IFR & enable interrupts.
                                } Error halts here.

                                KEYSERV:
0020  I  103      IN1           Read Keyboard (AGAIN even!)
0021  I  074      CPI           Is it null (ie: you let go of it
0022  D  FF      '11111111'      ... finally!!!) ?
0023  I  150      JNZ           }
0024  D  2D      L(ENDKEY)      } If so skip out.
0025  D  00      H(ENDKEY)      }
0026  I  054      XRI           Invert code for display, see
0027  D  FF      '11111111'      page 5 for comment.
0028  I  161      OUT30          Into Right Display*
0029  I  163      OUT31          Into Left Display*
002A  I  104      JMP
002B  D  20      L(KEYSERV)      Keep reading until you let go
002C  D  00      H(KEYSERV)      the fool keyboard!!!

                                ENDKEY:
002D  I  101      INO           (A-reg has desired enable code
002E  I  377      HALT          in bits 0-1, so don't bother
                                to use a literal.)

```

Further Notes & Comments: When ENDKEY is reached, it is because the accumulator was found to have "FF" (hex) - thus the two bits 0 and 1 in particular are "on" and will set the Interrupt Enable Flip Flop as well as clear the Interrupt Flag Register (IFR) setting things up for the next iteration of the whole program. Similar processing is used in the second sample program.

* The notes at addresses 0028 and 0029 refer to physical placement of the output displays. This program assumes (as do succeeding programs) that OUTPUT DEVICE 31 constitutes the left half of a line of 16 LED's and that OUTPUT DEVICE 30 is the right half. Together, a 16-bit display in the conventional sense (high order at left) is possible.

The form '11111111' is used here to indicate binary literal patterns. Refer to article ECS-3 for definitions of other conventions.

BINARY-CALCULATOR Program: With the completion of the keyboard and simple displays, it is now possible for the builder of an Experimenter's Computer System to consider potentially useful applications programs. As a further example of I/O handling and to give the outlines of a whole generic class of programs, a simple "binary calculator" of 16 bit precision is illustrated in this article. The program interprets the following commands, maintaining an ENTRY register and a SUM register in the program operation:

"E" key: Clear the ENTRY register, display SUM.
 "C" key: Clear the SUM register, clear display.
 "S" key: Subtract ENTRY from SUM, display sum.
 "A" key: Add ENTRY to SUM, display sum.
 "1" key: Enter Binary "1" digit (into low order, shift previous entry.)
 "0" key: Enter Binary "0" digit (also into low order shifting previous entry.)

The program as listed assumes the hardware keyboard definitions as found in this article - if you use a different keyboard, with different input coding, you will have to change constants at locations 22, 27, 2C, 31, 36, and 3B. Furthermore, if you do not like my choice of keyboard keys, feel free to pick your own and change the corresponding locations. In this listing, the notation C"X" means the character X's code. To operate the program with present hardware, enter it in bootstrap mode then place the computer in "run". Hitting any one of the 6 defined keys will then cause the program's corresponding routine to be executed with appropriate consequences for ENTRY, SUM and the display outputs. Now, the listing...

INTERRUPT:				
0000	I	006	LAI	The program wakes up when you
0001	D	02	02	hit a key. Disable Interrupts
0002	I	101	INO	and read the IFR.
0003	I	056	LHI	Entire program runs in page 0
0004	D	00	H(PAGE-0)	so set H and forget it!!
0005	I	012	RRC	Shift keyboard flag to carry.
0006	I	140	JTC	
0007	D	20	L(BINCALC)	Test keyboard flag and branch
0008	D	00	H(BINCALC)	to calculator if it is it!
0009	I	006	LAI	Otherwise, error similar to that
000A	D	00	00	handled by KEYBOARD-ECHO.
000B	I	161	OUT30	All "1"s to display 30 (compl.)
000C	I	163	OUT31	All "1"s to display 31 (compl.)
000D	I	006	LAI	
000E	D	03	'00000011'	Reset Int. Enable FlipFlop.
000F	I	101	INO	with output from INO
0010	I	377	HALT(HLT)	#1 Halt in list.

Data Definitions For BINARY-CALCULATOR:

001C = ENTRY-LO }
 001D = ENTRY-HO } Current ENTRY (16 bits)

001E = SUM-LO }
 001F = SUM-HO } Current SUM (16 bits)

BINARY-CALCULATOR, continued...

Here is the command interpreter routine, to continue the listing. It is a very simple-minded approach useful when a small number of commands is involved - it simply checks on each possible combination, only executing the routines if it finds the right code.

```

BINCALC:
0020  I    103    IN1      Read the character.
0021  I    074    CPI
0022  D    D3     C"S"
0023  I    150    JTZ      Should I subtract??
0024  D    50     L(SUBTRACT)
0025  D    00     H(SUBTRACT)
0026  I    074    CPI
0027  D    C1     C"A"
0028  I    150    JTZ      Or should I add??
0029  D    70     L(ADDER)
002A  D    00     H(ADDER)
002B  I    074    CPI
002C  D    C3     C"C"
002D  I    150    JTZ      Or maybe clear the SUM?
002E  D    9B     L(CLEAR-SUM)
002F  D    00     H(CLEAR-SUM)
0030  I    074    CPI
0031  D    C5     C"E"
0032  I    150    JTZ      Or just clear the ENTRY?
0033  D    90     L(CLEAR-ENTRY)
0034  D    00     H(CLEAR-ENTRY)
0035  I    074    CPI
0036  D    F1     C"1"
0037  I    150    JTZ      Can't calculate without data!
0038  D    A0     L(ONE)
0039  D    00     H(ONE)
003A  I    074    CPI
003B  D    F0     C"0"
003C  I    150    JTZ      who cares what data!
003D  D    A5     L(ZERO)      either will do...
003E  D    00     H(ZERO)
003F  I    006    LAI
0040  D    03     '00000011'
0041  I    101    IN0
0042  I    377    HLT(HALT)    OK dummy! why'd you press that
                                undefined key? Set interrupt
                                enable and quit.
                                #2 Halt in the list.

```

After the interrupt, you go to BINCALC - if the key pressed was "S" you in turn pass on to...

```

SUBTRACT:
0050  I    066    LLI
0051  D    1E     L(SUM-LO)
0052  I    307    LAM
0053  I    066    LLI
0054  D    1C     L(ENTRY-LO)
0055  I    227    SUM
0056  I    066    LLI
0057  D    1E     L(SUM-LO)
0058  I    370    LMA

```

Subtract low order of ENTRY from SUM, mindful of borrow result...

And of course save it!

BINARY-CALCULATOR, continued...

0059	I	060	INL	}	Subtract high order ENTRY from SUM with borrow input from previous.
005A	I	307	LAM		
005B	I	066	LLI		
005C	D	1D	L(ENTRY-HO)		
005D	I	237	SBM	}	And - as always - save it!
005E	I	066	LLI		
005F	D	1F	L(SUM-HO)		
0060	I	370	LMA		

Now after any operation, whether addition or subtraction, it is desirable to show some results. The following routine is reached after the subtraction, or by branching from other routines...

DISPLAY-SUM:					
0061	I	054	XRI	}	See note on page 5... must send inverted data to H.O. display.
0062	D	FF	'11111111'		
0063	I	163	OUT31		
0064	I	061	DCL		
0065	I	307	LAM	}	Fetch L.O. display from SUM-LO and invert and output it...
0066	I	054	XRI		
0067	D	FF	'11111111'		
0068	I	161	OUT30		
0069	I	006	LAI	}	Interrupt Enable Code sent out with INO... #3 Halt in the list...
006A	D	03	'00000011'		
006B	I	101	INO		
006C	I	377	HLT		

Another command option was addition. The following routine shows a 16 bit add operation in many ways similar to the subtraction above, but a bit more compact in principle (not actual due to jump at end) due to taking advantage of commutivity of addition operations...

ADDER:					
0070	I	066	LLI	}	Add the low order first
0071	D	1C	L(ENTRY-LO)		
0072	I	307	LAM		
0073	I	066	LLI		
0074	D	1E	L(SUM-LO)	}	With ENTRY in accum, address now points to SUM for save... But not for long...
0075	I	207	ADM		
0076	I	370	LMA		
0077	I	066	INL		
0078	D	1D	L(ENTRY-HO)	}	Add high order last with carry from previous.
0079	I	307	LAM		
007A	I	066	LLI		
007B	D	1F	L(SUM-HO)		
007C	I	217	ACM	}	Save - again with fortuitous lack of address definition.
007D	I	370	LMA		
007E	I	104	JMP		
007F	D	61	L(DISPLAY-SUM)		
0080	D	00	H(DISPLAY-SUM)		

The program continues on the top of page 20 with definition of the two routines CLEAR-ENTRY and CLEAR-SUM - written in an interlocking manner to share some common code, "CLEAR-EITHER". The separate routines set the address to be cleared - after which general purpose code is used.

			CLEAR-ENTRY:	
0090	I	066	LLI	Set addressability of ENTRY
0091	D	1C	L(ENTRY)	
			CLEAR-EITHER:	
0092	I	076	LMI	} Clear wipes out two bytes at address in L-register on entry...
0093	D	00	00	
0094	I	060	INL	
0095	I	076	LMI	
0096	D	00	00	} Then in a simple-minded manner resets interrupts with output of INO and goes to sleep. #4 halt in list.
0097	I	006	LAI	
0098	D	03	'00000011'	
0099	I	101	INO	
009A	I	377	HLT	
			CLEAR-SUM:	
009B	I	066	LLI	Set addressability of SUM
009C	D	1E	L(SUM)	
009D	I	104	JMP	
009E	D	92	L(CLEAR-EITHER)	And go clear it
009F	D	00	H(CLEAR-EITHER)	

And finally, the last set of routines in the program is the data input methodology. Basically, since this is a binary calculator, it was decided to input data on a bit serial basis - shifting each "1" or "0" key stroke into the ENTRY register as it comes. The high order bit is thus entered first for a number, followed by as many binary digits as required for its precision.

			ONE:	
00A0	I	046	LEI	} First set the digit "1" if you come here...
00A1	D	01	'00000001'	
00A2	I	104	JMP	
00A3	D	A7	L(SHIFTIN)	} Then jump to either routine...
00A4	D	00	H(SHIFTIN)	

			ZERO:	
00A5	I	046	LEI	} Set the digit "0" if you come come here instead...
00A6	D	00	'00000000'	

			SHIFTIN:	
00A7	I	250	XRA	Clear carry (vy important!)
00A8	I	066	LLI	
00A9	D	1C	L(ENTRY-LO)	} Fetch low order ENTRY to accumulator...
00AA	I	307	LAM	
00AB	I	022	RAL	Make room for new bit...
00AC	I	036	LDI	} Save bit for high order input...
00AD	D	00	'00000000'	
00AE	I	140	JTC	
00AF	D	B3	L(NOT-ONE)	
00B0	D	00	H(NOT-ONE)	
00B1	I	036	LDI	
00B2	D	01	'00000001'	

BINARY-CALCULATOR, continued...

NOT-ONE:

00B3	I	264	ORE	Add in new Low order bit
00B4	I	370	LMA	Save low order ENTRY
00B5	I	250	XRA	Clear carry...
00B6	I	060	INL	Fetch the high order
00B7	I	307	LAM	portion of old ENTRY
00B8	I	022	RAL	Make room for new bit
00B9	I	263	ORD	Add in shift out of ENTRY-LO
00BA	I	370	LMA	And save - always save!
00BB	I	054	XRI	
00Bc	D	FF	'11111111'	} Invert and send to display. defines left half (H.O.)
00BD	I	163	OUT31	
00BE	I	061	DCL	Point to ENTRY-LO.
00BF	I	307	LAM	Fetch L.O.
00C0	I	054	XRI	} Invert and send to to display low order this time
00C1	D	FF	'11111111'	
00C2	I	161	OUT30	
00C3	I	006	LAI	
00C4	D	03	'00000011'	As usual, set interrupt enable
00C5	I	101	INO	and clear IFR then go to sleep.
00C6	I	377	HLT	#5 in list.

This completes the listing of the BINARY-CALCULATOR program. The program can be "toggled in" to the system as it stands once you have completed the I/O devices of ECS-5 - but don't be afraid to try it out even before you get the I/O stuff going, for except for the detailed interactive operation, the program will run - missing pieces of your computer will give default states.

This program is written and listed in "absolute" machine code - there is no provision in the program for loading it at arbitrary addresses in memory. As a result, all those who plan to load it at different addresses in memory will have to adjust the various constants which reference addresses within the program.

As an exercise, you might try altering the program in several different ways:

1. Add new functions to the command interpreter on page 18 of this article. The addition of new interpreter code is straight forward - but you will have to supply the detailed functions of your extensions. An easy command to add would be that of exchanging the contents of SUM and ENTRY. Another might be to complement the ENTRY.

2. Look at the code from the standpoint of "optimization"... the minimization of memory and time of execution. I wrote this example with no large measure of thought on the subject of minimizing the size of code - as a result you should be able to find several ways of "improving" the memory and time efficiency of the program. For instance, keeping addresses around in unused registers might be a way of minimizing the number of "LLI" instructions - but you must be careful to guarantee that a CPU register is truly available.

3. Build yourself another output display register, for example, OUT32. Then re-write this program for a 24-bit calculation precision.

DEPARTMENT OF PROGRAM PATCHING:

Murphy's Law reigns! The description of the memory test program given the last issue (ECS-4, pages 8-10) had an unfortunate error in the (of all places) ERROR routine! If you inspect the code in detail you will find the following:

1. The high order bytes are never stored back into memory.
2. Register usage is mismanaged in that the TEST routine assumes that register L will not be touched by ERROR...ditto H

In order to set the record straight, the following is a corrected copy of ERROR which supercedes the listing on page -10- of ECS-4.

```

                                ERROR:
0080  I   336      LDL          Save L addressability
0081  I   345      LEH          Save H addressability
0082  I   066      LLI
0083  D   63      L(ECOUNT)
0084  I   056      LHI
0085  D   00      H(ECOUNT)      Add 1 to first ECOUNT byte
0086  I   006      LAI
0087  D   01      1
0088  I   207      ADM
0089  I   370      LMA
008A  I   066      LLI
008B  D   64      L(ECOUNT+1)    New address without zapping C flag.
008C  I   006      LAI
008D  D   00      0
008E  I   217      ACM          Add second byte with carry.
008F  I   370      LMA          AND SAVE SAVE SAVE!!!
0090  I   066      LLI
0091  D   65      L(ECOUNT+2)    New address, save C-flag
0092  I   006      LAI
0093  D   00      0
0094  I   217      ACM          Add third byte with carry
0095  I   370      LMA          AND SAVE SAVE SAVE!!!
0096  I   066      LLI
0097  D   66      L(ECOUNT+3)    New address without zapping C flag.
0098  I   006      LAI
0099  D   00      0
009A  I   217      ACM          Add fourth byte with carry.
009B  I   370      LMA          AND SAVE SAVE SAVE!!!
009C  I   363      LLD          Restore L addressability of TEST
009D  I   354      LHE          Restore H addressability of TEST
009E  I   003      RFC          Return if not too many errors (?)
009F  I   377      HLT          Quit if more than 232

```

As is usually the case in programming, there are alternatives. There is in particular, an alternative to this error incrementing routine which can be performed using the increment instruction, with returns executed as soon as no carry is indicated. As an exercise in programming, see if you can figure out such a routine. The answer will be found in the test program accompanying the ECS-8 design.

SOME PROGRAMMING NOTES - BITS AND PIECES:

How can you conveniently program a computer without an assembler or other automated program development tools? This is a problem of utmost concern to the individuals who assemble (note double meaning) their own computers along the lines of this series of articles - or based on other design concepts of a similar nature. One idea which is most useful in this area is the concept of a "symbol table" - a list of addresses at a known absolute location which is accessed by a "symbol" using appropriate subroutines.

First, what is the problem involved? Suppose that you have spent an hour or two "togglng in" a complicated program - only to discover that you made a mistake in writing your program on paper prior to entering it into memory - symbol X, a widely used variable in the program, is at the wrong location! Now, since X is used throughout the program, there is one obvious but tedious solution to the error - find every reference to "X" and change the address at that point in the program where "X" is referenced. In the Intel 8008 architecture, this is complicated by the fact that both the L and H portions of the address might appear - or one or the other if not both.

Now under these circumstances, you might be tempted to give up in frustration, pull the plug, power on again and re-enter the program. But suppose instead, that you had a bit of foresight and programmed symbolically using a symbol table and the following set of "service subroutines:"

LOADA: Loads the address of the symbol passed in register .
Output is in the content of H and L registers.
LOADAI: Loads the address of the symbol passed in register A,
with the current value of register B added as an index.
Output is in the content of H and L registers.
SETA: Allows changes to the symbol's address by returning
the place in the symbol table where the symbol's address is
located.

Suppose then, that you wished to store a result of an operation in symbolic location X. You would reference X as follows:

LLI	"x"	point to X
CAL	LOADA	define L,H from X's table entry
LMA		(A had result to be stored.)

Because the code of this little stretch of program is completely independent of the particular location of X - it depends only on the symbol table entry - to change the location of X involves only changing one item and not a myriad of references throughout code. The SETA operation would probably be performed once at initialization time for a large program - once the symbol is defined it probably will stay in the same place for most programs. The LOADAI - indexed address load - is useful as an extension to the concept by enabling a more powerful method of referencing with an index.

The same symbol table concept can be used to define symbolic JMP and CAL routines in much the same way.

OF INTEREST TO READERS:

At the time this article is being written (November 17 1974) a prototype for the design of article ECS-8 is up and running - in printed circuit board form. This design is a 1K by 8 bit memory page which can be placed at any one of 16 address locations in your Experimenter's Computer System. Since memory is used repeatedly, I have taken the time to make this design in PC form for ease of reproduction.... and all ECS-series subscribers can take advantage of the production PC boards, the second product being offered in support of these articles. At this time, I have no firm pricing information on production versions, but watch the next issue for more details.

At the time this issue goes to press, a new catalog, Catalog 3, is at the printers - with copies in the first lot expected any day. As new catalogs are printed they will be sent to subscribers along with regular issues. So if the catalog does not make it with the mailing of ECS-5, look for it in your ECS-6 issue.

A thought which has occurred in recent weeks regarding programming the ECS system, especially the more useful systems programs, is the following: Such programs could be distributed as sets of fully programmed ROM modules which can plug into the bus structure of the system. The price of such a product, including documentation and the chips mounted on a P.C. board would be in the \$50 to \$100 range. The next question is this: is there subscriber interest in such a product? I'll extend the subscription of the first reader to give me a note with thoughts on the subject, by one issue - with a drawing to determine who gets the extension in the event of identical postmark dates. A prime candidate for this treatment is the "IMP" (Interactive Manipulator Program) software which is being written now for ECS-7.

Carl T. Helmers, Jr.
Carl T. Helmers, Jr.
Publisher

The Experimenter's Computer System: Part 6

SERIAL I/O INTERFACE INTRODUCTION & ERRATA FOR
PREVIOUSLY PUBLISHED ARTICLES

by. Carl T. Helmers, Jr.

INTRODUCTION:

This article is the sixth number in the Experimenter's Computer System series. It continues the description of hardware and software begun in the earlier articles by providing information on the following topics:

Introduction to the ECS-6 Serial I/O Interface - the information found in this article includes the beginning of the technical description of the ECS-6 serial I/O interface design, with a discussion of the overall system description and definitions of interface signals. The design presentation will be continued in the next issue with detailed logic diagrams and related information.

Technical Updates and Errata. - a portion of this article is devoted to the correction of several technical and editorial errors in articles ECS-3 and ECS-5 previously published. Also included are several technical improvements on the original designs.

A Bit of Fun - The CATERPILLER is a simple demonstration program which uses the ECS-5 binary output display lamps to illustrate the operation of shifting bits - to the amazement of friend and family.

Reader's Reactions - a portion of this article is dedicated to notes and comments regarding the ROM software idea and other inputs from subscribers. The winner of the informal contest announced in ECS-5 is included in this section's information.

ANNOUNCEMENT OF A CHANGE OF FORMAT & TIMING:

Effective with the next issue of an article, the Experimenter's Computer System will become a monthly magazine. It is my intention to retain a format of one major technical topic per issue, with minor topics and departments included on a discretionary basis. Complex technical systems such as the ECS-6 design will in general be spread over one or two issues. All present subscribers will be mailed Volume 1, No. 1 under the new arrangement in January - with subsequent issues on a monthly basis for the term of the subscription. Hardware designs will continue to be numbered as in the past.

Carl T. Helmers, Jr.
Carl T. Helmers, Jr.
Publisher

Dec. 15 1974

CHANGE OF DESIGN CONCEPT:

As originally conceived, the ECS-6 design was to be a "tape controller" for the conversion of parallel CPU data into a serial format and vice versa. This conversion process is fairly general - especially if an "assynchronous" data format is used, generated by a "UAR/T" chip (Universal Assynchronous Receiver/Transmitter.) The design incorporates the UAR/T function for the serial/parallel and parallel/serial conversions. It also includes selection logic for four "channels" assumed to be one teletype device at 110 baud, plus from one to three serial tape recorder data interfaces of the ECS-2 design or equivalent. A binary counter is used to select data rates for the conversion from 110 baud to 1760 baud, programmable with information defined by the CPU and software. The low end of the frequency range was selected as 110 baud in order to achieve teletype compatibility, thus extending the device concept beyond the original idea of a serial tape interface. A teletype current loop output interface and brush contact input switch are assumed, as used in the Model 33 Teletypewriter.

BLOCK DIAGRAM OF THE SERIAL INTERFACE:

On page 3 of this issue (opposite) is a block diagram of the Serial Data Interface design ECS-6. This diagram outlines the major functional sections of the device and provides a referencepoint for the discussion which follows below. The detailed logic diagram of this hardware will be published in the next issue.

UAR/T DEVICE:

The heart of the serial/parallel/serial conversion technique used in this design is an LSI UAR/T chip. The basic circuit definition and pinouts of this 40-pin package IC are fairly universal, with several different manufacturers making pin-compatible devices. The prototype was built using a Standard Microsystems COM2502 device which cost approximately \$13.50 new in quantities of one. Other manufacturers of this type of chip include:

Signetics (2536), T.I. (TMS-6010)

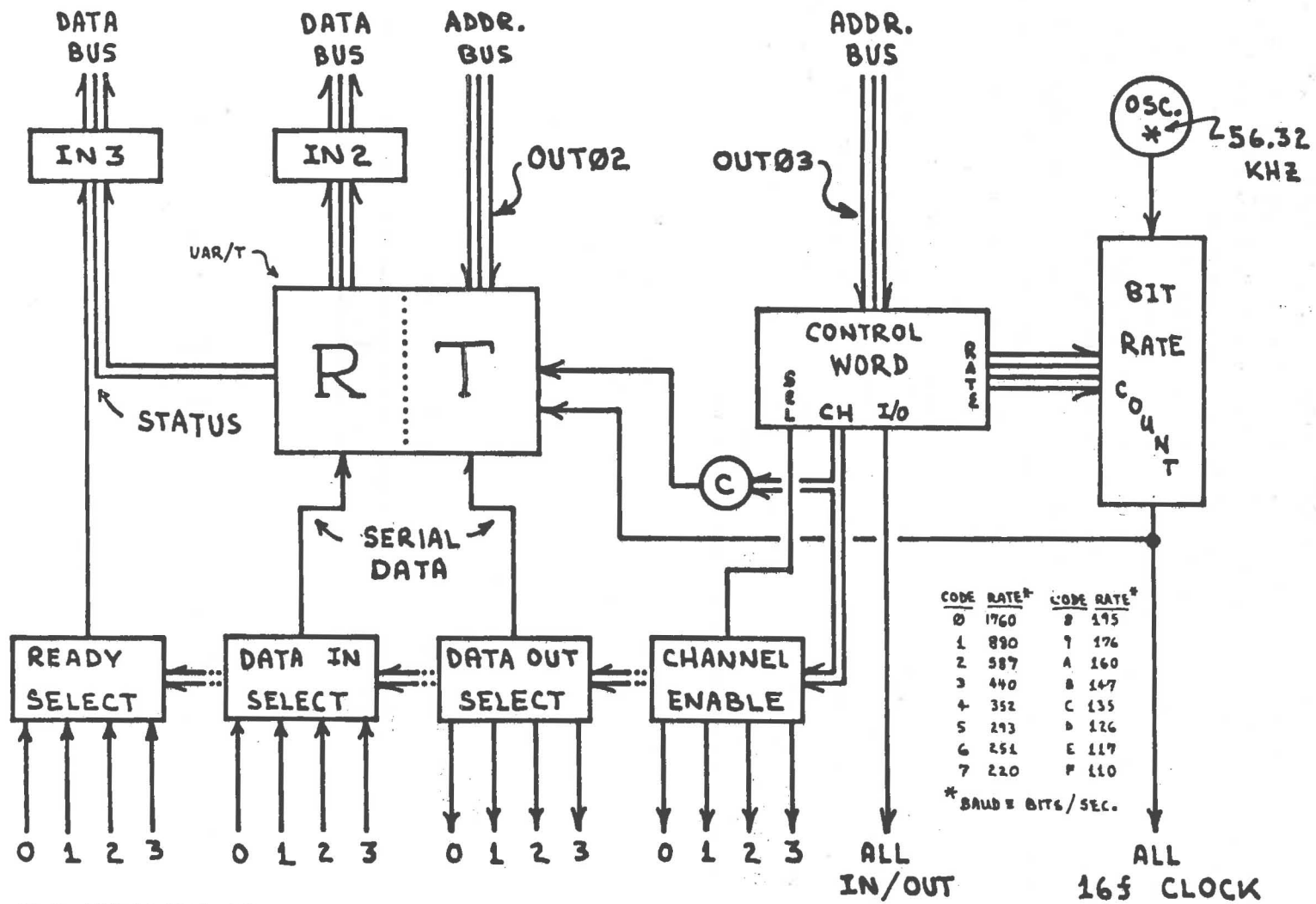
The Signetics documentation lists several other numbers as pin-compatible and presumably electrically compatible including "AY-5-1012", "TR-1402A" and "SI757" - with identification of manufacturers left unspecified.

The UAR/T device is itself divided into two functional sections. In the block diagram, these are labelled "R" for "receiver" and "T" for "transmitter." These sections are independent in operation, although parity and word length settings are in common. The receiver can analyze one data stream at one clock rate while the transmitter is sending out a second data stream at a second clock rate.

Internally, each section of the UAR/T has buffering via a register of 8 bits, with a second 8 bit shift register used for the serial/parallel (receive) or parallel/serial (transmit) conversions. On output in the ECS-6 design, the first write operation of a series places data into the "transmitter buffer register" - and the UAR/T immediately transfers this to the "transmitter shift register" to begin the first character output. The CPU then writes a second character to fill the buffer again, and enters an interrupt response mode character by character until all data is transferred. After the last character is sent, one final interrupt occurs to indicate that the last character was completed and the tape can be turned off.

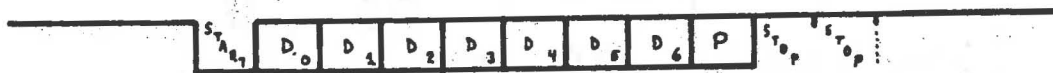
EXPERIMENTER'S COMPUTER SYSTEM - SERIAL DATA INTERFACE (ECS-6) BLOCK DIAGRAM

by Carl T. Helmers, Jr.

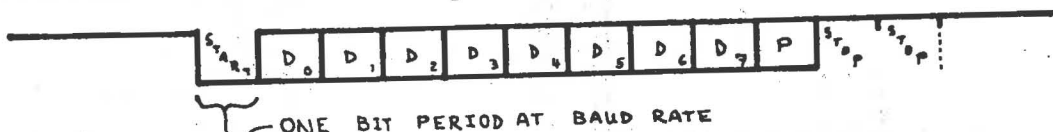


For input, the start bit of the data received begins UAR/T operation for a character of information. Once the tape motion has begun and the tape start up transients have completed, the UAR/T will begin analyzing the input bit stream looking for the "1" to "0" transition marking the start of a character's information. The UAR/T includes within it the logic needed to discriminate against short noise spikes and other spurious start pulse conditions - it simply checks to make sure that the data is still low exactly one half of a bit time later. The reason for putting in a clock frequency 16 times the data frequency (see below) is so that the UAR/T can digitally count down the time between the leading edge and the center (8 clock pulses) of the start bit. After the start bit, information is shifted into the "receiver shift register" generating the received parity information as it goes along. At the end of the "n" bit (7 for TTY, 8 for normal tape data) string, the parity found is checked against the parity bit incoming - and a parity error is detected upon mismatch. The CPU interrupt for receiving data is generated by the "receiver data available" status flag's rising edge triggering a one-shot. The CPU must respond by reading the data (which also clears the buffer in the ECS-6 design and turns off the data available flag.) If - due to a software bug or other flaw in the program - the CPU has not responded to the character within the time it takes to read another word from the input stream, a "receiver over run" error condition is indicated - and the appropriate error flag is set.

The UAR/T data format is a generalized asynchronous serial format in which a "start" bit transition from logical "1" to logical "0" for one bit period cues the start of a character - followed by several data bits, an optional parity bit and one or two stop bits. The stop bits transmitted on output simply represent the minimum inter-character spacing for valid data - there is no need to transmit again immediately. In the specific design of ECS-6, the general programmability of the device was intentionally limited to two data formats. These are the standard Teletype format used for channel 0 output and input,



And an 8-bit serial data format used for channels 1 2 and 3 of the output device, intended to interface with serial tape recorders:



In the block diagram of page 3, a circle labelled "C" to the right of the UAR/T symbol represents control logic used to choose the length of the data bit field as "7" or "8" depending upon channel. The high order bit of data is unused for teletype output and input.

CONTROL WORD / STATUS WORD:

A basic tool for controlling the Serial Data Interface module is provided by the "IN3" instruction of the Intel 8008 CPU used in ECS-3. This instruction is used to output a "control word" and read a "status word" during I/O operations. The function of this operation is to send out the current accumulator content to the Control Word and replace it by the current condition of the Status Word. The basic interactive programming sequence for control and status checking is thus the following:

n	LAI	load the accumulator
n+1	"? ? "	with control word data
n+2	IN3	output control, read status
n+3		beginning of routine to analyze status

This sequence is used whenever the control word is to be changed and/or the status is to be checked. The basic data definitions for the control word are as follows:

Bit 0 - SELECT. This bit has two purposes in the system:

- a. In its "0" state, it presents a "Master Reset" to the UAR/T chip to initialize I/O operations.
- b. In its "1" state, it acts to enable logic in the device being addressed by the CHANNEL code - serving as a "motor on" signal for tape recorder I/O for example.

Bit 1 - IN/OUT. This bit has the function of selecting the direction of data transfer. It is used to control whether a UAR/T buffer read (input) or buffer write operation is performed by IN2(OUT03). It is routed to all devices of the system. For the tape devices this bit selects the source of the CPU interrupt used with this hardware, since only input or output but not both simultaneously is supported for these devices. "1" Indicates input and "0" indicates output in this bit position.

Bits 2 and 3 - CHANNEL. These two bits select the "Channel" being used for an I/O operation. The following assignments are assumed by this hardware design:

Channel 0 ("00") is the Teletype (TTY)
Channel 1 ("01") is tape
Channel 2 ("10") is tape
Channel 3 ("11") is tape

Bits 4 to 7 - BAUD RATE CODE. These four bits are used to form a single hexadecimal digit, used to program the binary divider which sets data transmission rate. These codes are listed with corresponding frequencies in the block diagram on page 3.

The basic data definitions of the status word input are as follows:

Bit 0 - PARITY ERROR. This status bit is set to logic "1" by the UAR/T receiver section to indicate that the received parity bit does not agree with the parity information regenerated by the receiver using the data bits as received. Parity is generated by the UAR/T transmitter section in this design and as a result the parity error condition indicates an error between transmission and reception. A basic assumption in this error detection technique is that the probability of one error is low - and that therefore the probability of two simultaneous errors is miniscule due to the multiplicative properties of probability measures. It turns out that parity will detect any odd number (eg: 1, 3, 5, etc.) of simultaneous errors - but will completely miss an even number of errors. More sophisticated error correction and detection techniques involving multiple bit codes provide additional "redundancy" in the information and the ability to correct single and even multiple bit error conditions. In the ECS-type of system, a "brute force" technique of using multiple copies of the data involved could be implemented in software if an "air tight" guarantee against errors is required to alleviate data loss worries.

Bit 1 - OVERRUN ERROR. It is possible to consider the possibility that a software bug or other intervention might cause the CPU response to an input interrupt to exceed the maximum time allowed by the data transmission rate. This bit of the UAR/T status output is provided to indicate such an "overflow error" condition. In the bit rates available in the specific design shown in this publication series, the CPU response must range from a 100 millisecond maximum for the 110 baud TTY format, to a maximum delay of 6.82 milliseconds for 1760 baud information in the general 12-bit serial format - useful in high speed magnetic recording media. The 6.82 ms delay gives the CPU an equivalent of 340 five-state (20 microsecond) 8008 instructions to execute without threat of overrun.

Bit 2 - FRAMING ERROR. This status bit is set to logical "1" by the UAR/T receiver section to indicate an error in the format of received characters. Such format errors are defined as an invalid stop bit following the parity bit in the serial data.

Bit 3 - END OF CHARACTER. This status bit is used to indicate that the transmitter section has finished transmission of a character. It is tested after the last character has been written, so that software can determine when it is safe to turn off the unit. During transmission it also serves as the source of interrupt pulses.

Bit 4 - TRANSMITTER BUFFER EMPTY. This status bit is a state level which indicates to the CPU software that the output buffer can be written into. This status bit is used during the beginning of output of a block of data to tape - and prior to each character transmission to the Teletype device.

Bit 5 - RECEIVER DATA AVAILABLE. This bit is set when an input character has been completed and transferred to the receiver buffer register. The rising edge of this signal is used to cue the interrupt which drives character input software, and it is reset when the CPU responds with the IN2 operation code in this design.

NOTES CONCERNING UAR/T STATUS OUTPUTS:

1. The error condition bits are reset by the "Master Reset" signal, and in this design, this corresponds to the "unselected" state of the system (bit 0 of the Control Word being zero. See page 5.)
2. When the teletype channel is selected, bits 4 and 5 must be tested after an interrupt to determine the source. When other channels are selected, the IN/OUT bit masks one or the other of the possible sources for an interrupt. This rules out a direct interleaved I/O from one tape to another using the receiver and transmitter sections simultaneously, with the CPU monitoring. It is intended that this hardware be used with blocked information transferred to and from CPU buffers of arbitrary length.

Bit 6 - READY. This status bit is provided in this design so that software can test a "ready" line associated with the devices connected to the system. For the tape units, this line is to indicate the end of motor turn-on transients and the beginning of data transfers. For the teletype, this line will initially be unused - but may eventually be wired to the "local/on line" switch on the front panel of a model 33 Teletype. (Via suitable sensors).

Bit 7 - Unassigned at present.

DATA WORDS:

The interface design uses the "IN2" instruction of the 8008 CPU as decoded by the ECS-5 hardware to act as the data transfer mechanism. The potential of this instruction is to exchange the current content of the accumulator ("A" register) with the content of an I/O device's data. The effect of the IN2 operation in this design depends upon the state of the control word defined above on page 5, as follows:

1. If the Output selection is made (Control word bit 1 is "0") then the current accumulator content is written into the Transmitter Buffer Register.

2. If the Input selection is made (Control Word bit 1 is "1") then the Receiver Buffer Register is read into the accumulator, and the UAR/T is acknowledged to prevent an over-run error condition.

In the software for data transfers, the direction of transfer must always be initialized in the Control Word (using the "IN3" instruction as described on page 5) before the actual data transfer takes place with "IN2". For the software which drives the tape devices, this can effectively be done prior to the beginning of I/O transfers for a large block of data. For the software interfacing the teletype, this must be done for each character after decoding the source of the interrupt (receiver or transmitter) using the status bits read by an "IN3". Also, since the status bits can only be read at the same time as a control word is written, it is assumed in this design that an RAM location will be reserved in software for the "current" internal ly maintained value of the Control Word bits, for pre-loading the accumulator prior to the IN3 operation. The illustration on page 5 shows an "LAI" instruction - which is fine if the "current" value of Control Word bits is always maintained in the location "n+1" and no where else. Other instruction sequences could be used however to define the accumulator depending upon programming strategies.

DEVICE CONNECTIONS:

The bottom edge of the block diagram on page 3 shows the general interface signals to the individual output units which may be selected by the Control Word "CHANNEL" field. These signals are as follows:

READY - Each device is assigned a ready line input to indicate its status to software, as described above in the description of the status word format.

DATA IN - This signal is the serial data input in the UAR/T data format as received from the Tape Interface's de modulator . In this design, it is assumed to be in the 12-bit format shown on page 4 for channels 1 to 3, and in the 11-bit teletype format for channel 0.

DATA OUT - This signal is the serial data output of the UAR/T in either the Teletype or tape medium format as selected by the Channel coding. A multiplexor is used for routing so that serial bits are not sent to unused channels - a teletype listening to high speed bits would get confused to say the least.

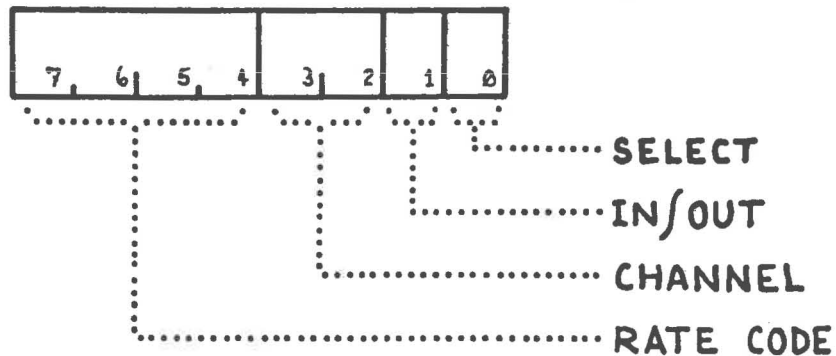
CHANNEL ENABLE - This signal is sent to each device when its channel is selected by the control word bits 2-3 and the control word select bit is "1".

IN/OUT - The Control Word bit is sent to all the devices, in order to select the logical direction of transfer in the modems used. (For TTY, this bit can be ignored.)

16-F CLOCK - This signal is a square wave clock at 16 times the bit rate for data, used by the UAR/T chip for its timing, and sent out to modems of the ECS-2 design to replace the local phase generation clocks. For serial I/O designs which do not need clock synchronism, this signal can be ignored.

CONTINUATION OF THIS DESIGN DESCRIPTION IN THE NEXT ISSUE: This discussion has covered the general outline of the ECS-6 design concept. The specific details will be covered as the major subject of the next article's content, the first in the monthly format of this series. Included in the next issue will be details of both the hardware logic for this design and its control using Intel 8008 software. The IMP program which will be described subsequently will assume the ECS-6 Tape Interface is available, since it will be used for bootstrap IPL of IMP and all subsequent software generated for the system. (For those unfamiliar with the term, "IPL" means "initial program load" - the process of automatically (as much as possible) entering software from offline storage devices. In the ECS-series type of computer based on an 8008 with manual memory access, a short program is entered to read the first block of data, followed by execution of the program just read into the machine - which in turn completes the definition and may even "zap" the hand built routine with portions of the final program load.)

CONTROL WORD FORMAT



TECHNICAL UPDATES AND ERRATA:

ECS-3 Timing Error:

The information on instruction execution time given in section 2 of the manual is off by a factor of 2 systematically. The times listed are exactly one half of the correct values. The following table shows the necessary conversions for each possible instruction time:

3 state instructions take 12 microseconds	@500 Khz.
5 state instructions take 20 microseconds	@500 Khz.
7 state instructions take 28 microseconds	@500 Khz.
8 state instructions take 32 microseconds	@500 Khz.
9 state instructions take 36 microseconds	@500 Khz.
11 state instructions take 44 microseconds	@500 Khz.

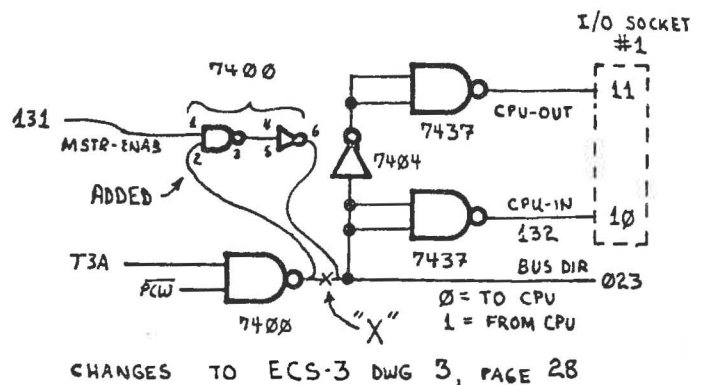
In order to provide an explicitly noted correct value for the instruction timings of all 8008 instructions and a quick reference listing of all operation codes (functional duplications omitted) this article's outside back cover is printed with an alphabetical (by mnemonic) listing of all instructions except the IN_x and OUT_{xx} codes. The listing includes mnemonic designation, operation code, the number of bytes required, the number of execution states, and the time in microseconds required for execution. The inside back cover completes the listing providing notation of all the I/O operation codes with space for comments regarding your own system's use of the codes.

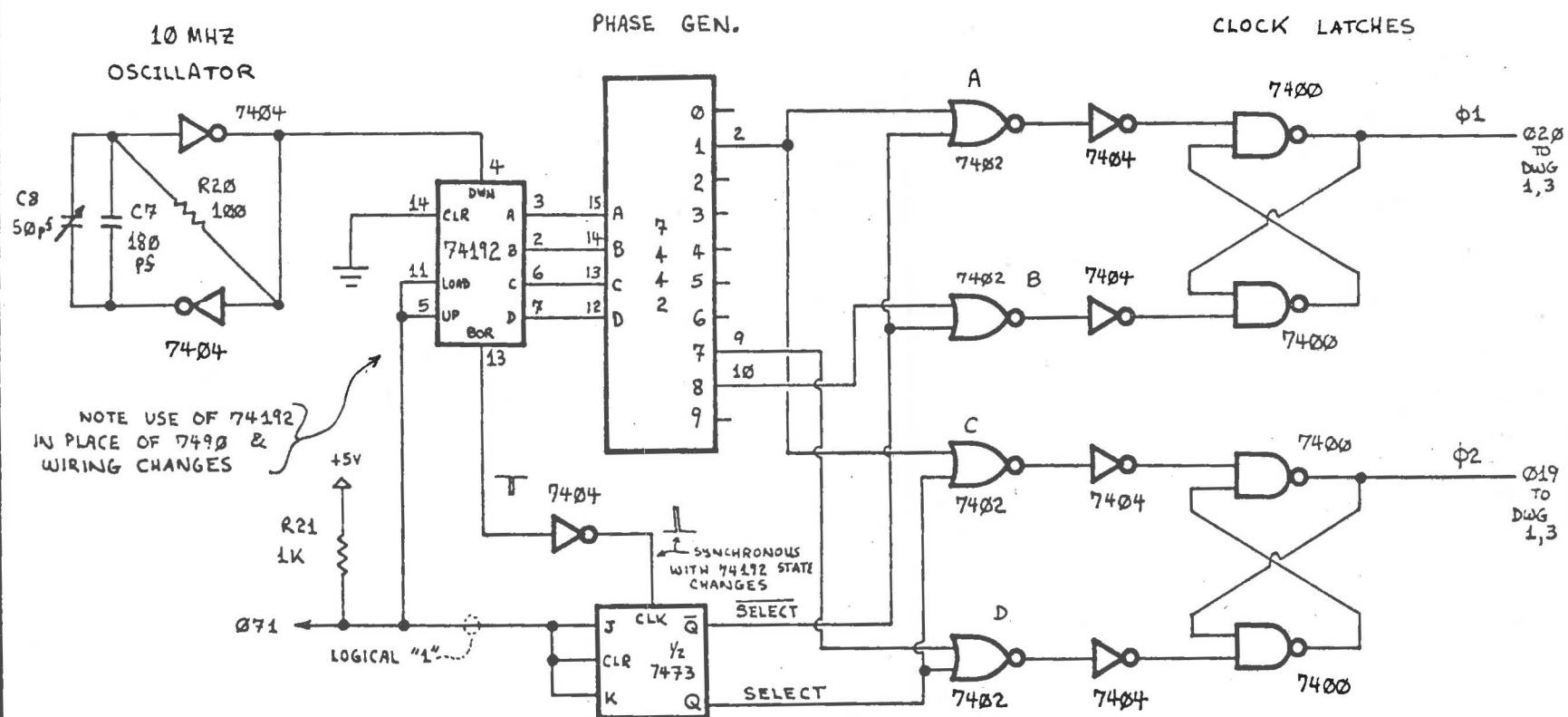
ECS-3 Clock Design Improvement:

Page 11 of this article contains a revised version of the ECS-3 drawing #4, clock generation logic. The improvement in design is the use of a 74192 counter in place of the 7490 used in the original version. This change obtains completely synchronous (within gate propagation delay tolerances) operation. With the previous design, it was possible - given random starting conditions - to lock up the clock into an erroneous counting/waveform state due to glitches in the 7442's output while the 7490's state change propagates asynchronously. The severity of the problem depends upon the particular 7490 and 7442 IC's used due to variations in propagation delays with individual circuits. By replacing the asynchronous counter with the synchronous 74192 and using the "borrow" output to toggle the flipflop the problem is eliminated. Other aspects of the clock design are unchanged with this improvement (see the diagram enclosed.)

ECS-3 Bus Control Logic Correction:

The bus control logic of the ECS-3 design, drawing 3, was found to be in error during the checkout of ECS-5. The state in which the CPU has been interrupted and normal bus control should be overridden by the interrupt





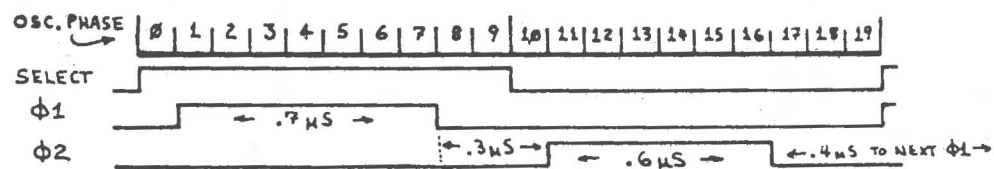
EXPERIMENTER'S COMPUTER SYSTEM

by Carl T. Helmers, Jr.

ECS-3 DWG. 4 : CLOCK LOGIC

REVISED: DECEMBER 1 1974

(REPLACES PAGE 29 OF ECS-3)



CLOCK TIMING
(NOM. 2.0 μ S PERIOD)

condition incorrectly allowed a bus enable during the interrupt "jam" cycle. This problem did not manifest itself until the bus loading had been expanded with the ECS-5 hardware and it was found that the CPU would occasionally lock in an erroneous state (for instance, during the course of an overnight memory test program run.) The logic in the upper right hand corner of drawing 3 on page 28 should be changed as shown in the inset on the previous page of this article. The new logic added is a functional "AND" of the normal CPU bus enable signal with the interrupt override signal, the MSTR-ENAB signal. A single 7408 gate section could be used here, but the NAND form is shown to take advantage of spare 7400 sections.

ECS-3 RST Instruction Omission:

The principles of operation section of ECS-3 omitted reference to the single byte call instruction, RSTn (for "restart.") The following information is an additional section which would be inserted in ECS-3 on page 55.

2.2.4.3.2 RESTART INSTRUCTION (RSTn)

The RSTn instruction is used to call a subroutine addressed at locations 0, 8, 16, 24, 32, 40, 48 or 56 depending upon the value of "n" (0, 1, 2, 3, 4, 5, 6 or 7 respectively.) Its operation is identical to a CAL instruction - the current program counter is pushed into the stack and execution continues at the target address picked by the code. It takes only one byte however.

Mnemonic: "RSTn" (1 byte)

where: "n" picks the nth 8-byte subroutine starting at location 0.

Operation Code: "0n5"

where: "n" is the subroutine address code 0 to 7 from the following table:

Code	Address	Code	Address
0	0000	4	0020
1	0008	5	0028
2	0010	6	0030
3	0018	7	0038

Binary Format: "00 nnn 101"

where: "n" indicates bits of the subroutine address code.

Timing:

5 states, 1 cycle

(20 μ s @500Khz)

Condition Flags: Unaffected.

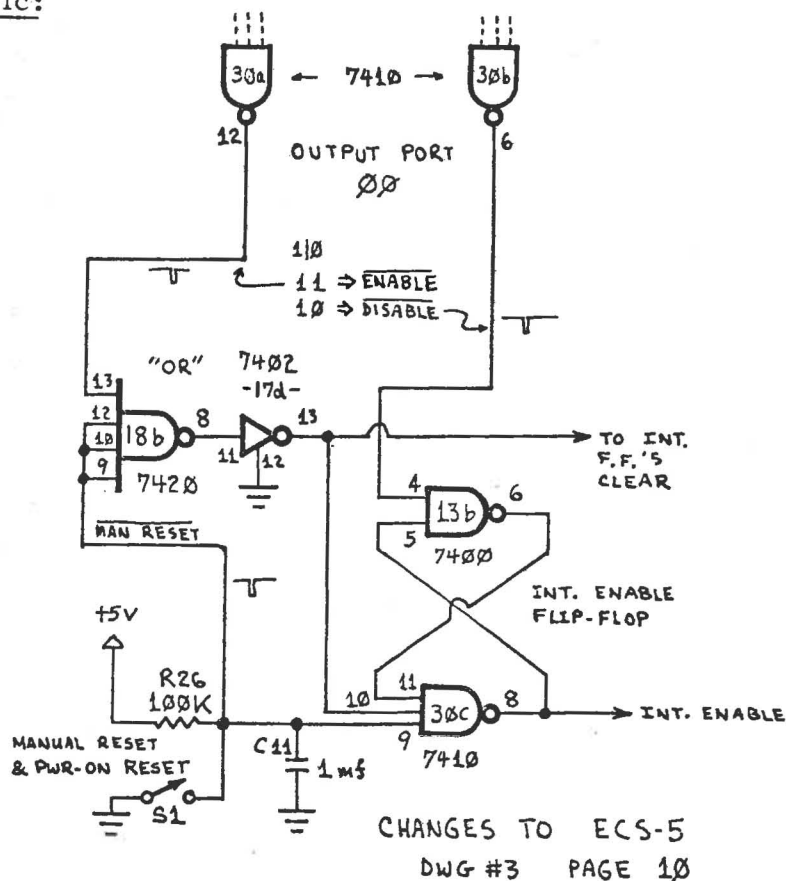
NOTE: See comment below re error on page 3 of ECS-3 in the statement of the possible branch targets of an RST instruction.

Miscellaneous ECS-3 Errata:

1. The Greek symbol " ϕ " used to represent the clock phases was omitted in textual references to ϕ_1 and ϕ_2 . This error is extensive in pages 4 to 8 of the theoretical discussion of operation, and is also found on pages 10 and 21 of the ECS-3 manual. A similar error occurs once on page 5 where the symbols " α_0 " and " α_7 " should have been noted in the last paragraph.
2. On page 3, the correct decimal addresses for the RST instruction are 0, 8, 16, 24, 32, 40, 48 and 56 corresponding to the RST0 through RST7 codes.
3. On page 6, section 1.3.1.3, "T1T" should read "T1A".
4. On page 6, section 1.3.1.4, "-911 volts" should read "-9 volts."
5. On page 14, CT3 should be added to the output of I/O socket #1 for use in the I/O controller, as noted in ECS-5.
6. On page 36, second paragraph, "is defined the" should read "is defined by the." In the fourth paragraph, same page "states add their" should read "states and their."
7. On page 37, table 4: The PCW and PCC cycle codes were switched by the typist. PCW's code should read "11" and PCC's cycle code is "01".
8. On page 38, section 2.1.6, "CPY" should read "CPU."
9. On page 40 at the top, "Arthitecture" should read "Architecture."
10. On page 43, first paragraph, "confied" should read "confined." On the same page at the bottom, item 3, "The In" should read "The INx".
11. On page 45, table 6, mnemonic XR explanation should read "Exclusive OR (XOR)."
12. Page 46, the Carry Flags heading, first line following should read "True implies operand greater than accumulator."
13. On page 53, a general comment: The discussion of the stack mechanism should have included mention of the maximum size available for calling nested levels of subroutines. The stack mechanism of the 8008 is 7 levels deep (ie: 7 program counter states can be saved) thus a maximum of 7 CAL or RST instructions (or conditional variants which are executed) can be performed in a series without any intervening RET instructions without losing information.

NOTE: This completes the errata and technical update information on ECS-3 as of the time of publication of this article. In future articles, as additional corrections are identified, information will be published in a similar format.

The buffering inverters labelled 14f and 15a and 15b in ECS-5 drawing #1 were inserted without considering the effect on pinouts in the diagram. To achieve a logical mapping of device code to select line invert the order of pin numbering for each group of 8 7442-generated bus enable or select signals. Thus pin 1 of IC 7 should be thought of as "7" not "0", and pin 9 should be thought of as "0" not 7 - with corresponding changes throughout. The reason for inserting the inverters in the published version is to normalize loading of the address lines - without inverters, there would be 5 TTL loads on those three lines in this subsection of the system, with only 1 load each on the other high-order address lines.



THE "CATERPILLER" - AN APPLICATION PROGRAM :

The following program was written to demonstrate an extended precision shift operation using the "rotate" instructions - which shows the result dynamically on the 16-bits of the binary display devices described by ECS-5. The result of this program is a moving pattern of lights in the display. The program is a simple one which first defines the initial data in registers D and E, then enters a loop which includes the shift operation, output of the result, and a time delay...

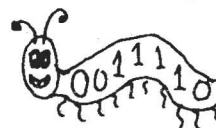
```

START:
0000  I    250  XRA      Clear initial carry.
0001  I    036  LDI      Define the first word.
0002  D    FF   "11111111" - all lights out.
0003  I    046  LEI      The caterpillar is a set
0004  D    C3   "11000011" of four "on" lights.

WALKLOOP:
0005  I    304  LAE      Fetch right byte
0006  I    022  RAL      Rotate left into carry, save
0007  I    340  LEA      and save the bits.
0008  I    303  LAD      Fetch left byte.
0009  I    022  RAL      Rotate it left too,
000A  I    330  LDA      and save it.
000B  I    163  OUT31    Write the left lamps.
000C  I    304  LAE      Get the right value.
000D  I    161  OUT30    Write the right lamps.
000E  I    026  LCI      Define delay time constant.
000F  D    C0   19210   A typical value.

DELAY:
0010  I    307  LAM      Use
0011  I    307  LAM      several
0012  I    307  LAM      longish
0013  I    307  LAM      delay
0014  I    307  LAM      instructions
0015  I    307  LAM      to
0016  I    307  LAM      stretch (s t r e t c h)
0017  I    307  LAM      out
0018  I    307  LAM      the
0019  I    307  LAM      loop.
001A  I    021  DCC      Decrement the counter.
001B  I    110  JFZ      and keep looping
001C  D    10   L(DELAY) Until
001D  D    00   H(DELAY) all
001E  I    104  JMP      done
001F  D    05   L(WALKLOOP) with
0020  D    00   H(WALKLOOP) realtime wait.

```



Once the program has been loaded using the bootstrap mode of the CPU its operation is begun by pressing any key on the keyboard to generate an interrupt which starts operation. The "on" bits will then march like a caterpillar through the display. Adjust the speed by changing the value of location 000F.

READERS' REACTIONS:

In the last issue an offer was made to extend the subscription of the first individual to write concerning a proposal to make software available in ROM form to subscribers. The winner - the only individual with a November 28 1974 postmark received - was James Fry of Toledo Ohio, whose subscription is now extended by one issue. The essence of his comment on the subject is this: standardize an audio tape data format and express programs using that format with cassettes for handling convenience.

George Fisher of Staten Island N.Y. sent along a copy of an article recently published in "PCC" (Peoples Computer Company, Box 310, Menlo Park Ca. 94025) concerning home-oriented microcomputer systems. The essence of the article's message is the desirability of cheap swappable ROM modules as the software customization medium for mass produced computers.

Marshall Horwitz, of La Mirada California sent along the following comments: "I feel the price of \$50 to \$100 is way out of line. ... Have you ever given any thought to writing a compiler program for ECS-3?" To the price comment, I can only reply that ROM's cost a fair amount in small quantities at present, with the Intel PROM's (eg: 1702) storing 256 bytes for approximately \$70 in small quantities for example (unprogrammed.) I think that if the re-programmability feature is sacrificed (as would be possible in a production program product for home computer builders) the fusible link type ROM's could be offered somewhat less expensively. The price of programming - whether with your own code or standards - would be an added amount to the basic hardware cost. As to compilers, the 8008 CPU has been around for some time now, with Intel itself pushing "PL/M" for large users. An obvious starting point for compilers is a simple interpretive language design. No homebuilt microcomputer constructed on a minimal budget will be able to handle much more. There is much instructional value re programming of the machine in a compiler project, and I am thinking now of devoting several issues in 1975 to such an interpretive compiler software system.

Donald Senzig of Union Grove, Wis. points out that a firm called "SCELBI Computer Consulting Inc." offers an ROM product whose price and description suggest the Intel 1702. I have seen a copy of their brochure and note that they do offer an 8008 kit product as well, which may be of interest to subscribers.

Gary K. Berkheiser of Bristol, Pa. sent along a thoughtful letter of considerable length, essentially endorsing the ROM idea as a useful one. He expressed considerable interest in the logical candidates for use of ROM's - the systems software needed to run the computer. Given a reasonable design for a tape interface, keyboard, etc. it is only necessary to program certain low level utility subroutines once - feeding it general and specific parameters whenever a program must do such I/O. These routines are used over and over, and their inclusion as ROM's will be useful. Mr. Berkheiser continues with several additional comments, including the following:

"... Your documentation of the Keyboard Echo and Binary Calculator programs is done quite well. It made them both much easier to read and a lot more enjoyable. But why jump from Hexadecimal to Octal and back again. I'm conversant in both counting systems but there may be some subscribers who are not and could become confused. Why not pick one system and stick with it. Hex seems to lend itself quite well as two digits represent eight bits of data instead of two and a half digits for octal...."

Mr. Berkheiser has raised a point which has been somewhat of a bone of contention between myself and my brother Peter for some time - since I began looking around for a means of expressing programs for the 8008 architecture and came up with the system used to date. Peter's argument is to use octal for everything, claiming with some truth that octal is an easier system to calculate mentally. AFTER ALL - the octal addition/subtraction tables (I doubt many people regularly multiply or divide in hex or octal!) are subsets of the usual decimal tables every civilized person learns to use early in life. But for a machine with the 8008 architecture, this runs directly into a major problem: the quantum of data is not divisible into three bit groups. The result, if 8-bit quantities which respect the H/L address division are used, is a crazy "pseudo octal" which counts up the low order three digits to 377, then carries over to 1000. The problems of converting mentally the pure octal output of a modified PDP-8 assembler for the Intel caused my associate Chris Bancroft to coin the word "Intelese" for this form of notation and to write an HP-65 program (which barely fit like a chinese puzzle into 100 keystrokes) to do the conversion for him in his work on intel-controlled industrial equipment. To contrast the notation systems for addresses, here is what happens as the program location goes from page 0 to page 1 in three systems:

<u>Intelese</u>	<u>Octal (pure)</u>	<u>Hex</u>
000000	00000	0000
:	:	:
000376	00376	00FE
000377	00377	00FF
001000	00400	0100

Note the impact of page boundary (after octal 377, hex FF) on the addressing sequence expressed in 8-bit Intelese quanta - the page boundary shows, at the price of losing a natural octal sequence. In octal, the page boundary is in the middle of a digit, so manually programming 8-bit words is difficult. But in hex, both the natural arithmetic sequence and a page bound significance between digits are achieved. It has never ceased to amaze me why manufacturers of machines with 8-bit (or 16) bit data quanta insist upon using octal. This is the case for instance in both the Data General NOVA and the Intel products. For some perverse reason many manufacturers insist on octal - maybe they are just trying to be "different" from IBM, which invented the use of hexadecimal notation in computing applications with the 360 series of machines. I am interested in providing readers with a method of "on paper" program expression which best fits the interests of convenient programming. If I take Mr. Berkheiser's suggestion (to use hex exclusively) the impact will be as follows:

1. All data expressions will be consistent with program operation code representations - at the price of eliminating the relation which which octal digits have to the internal format of 8008 instructions.
2. A new set of operation codes must be generated and used, expressing the same information as two hex digits instead of three numbers (two true octal, one high order quartal digit.) This eliminates certain mnemonic tricks useful for remembering instructions - for instance remembering the "3ds" (d is destination, s is source) form of the load instruction is easy and can be based on a mental algorithm rather than rote memorization needed with the hex form.
3. A problem disappears (which has not been mentioned previously in these pages) in the design of the IMP (Interactive Manipulator Program) now in progress. The interactive sequence required becomes much simpler if effectively only one base is used - there is no need to distinguish between "D" and "I" formats in entering or manipulating memory data with the program.

If you are interested in adding your views to this forum, on the program representation topic or other items of concern, drop me a line. I will not promise to include every comment made, but a selected few will appear in subsequent issues. To summarize the program representation question, the following courses are possible ...

1. All octal notation, consistent with machine op code structure, but awkward with 8-bit byte machines such as the 8008.
2. The notation which has been used in these articles to date, hex for addresses and data, numerical (effectively octal) strings for op codes which reflect the internal divisions of instruction bytes.
3. Pure hexadecimal as described above.

If you have comments, the deadline for inclusion in the next issue (due January 31 in the mail) is January 10 (plus or minus several days.)

A NOTE CONCERNING MEMORY BOARDS:

The price of the 1K memory array design, ECS-8, in PC board form will be \$19.00 fully drilled and \$14.00 undrilled plus postage. I have several boards on order at the time this article goes to press, and expect to contract production of additional boards according to demand. Advance orders can now be accepted - the boards use eight 2602 chips, two bus interface 8T09's, two 7404's, one 74154 and one 7400 - with two socket positions for data and address interfacing via DIP header plugs. If you order in advance I will ship the boards as soon as manufactured (once my small supply is gone) - with minimal documentation. The article to be published in early february will provide the detailed information.

OP CODE REFERENCE TABLES:

On this page are listed the 32 possible input and output op codes for the 8008 computer, in an "instruction" format of three digits, as described in ECS-3. All these instructions take 24 microseconds (OUTxx) or 32 microseconds (INx) at a 500 KHz clock rate.

On the last page (outside cover, page 20) is a complete listing of all the non output/input instructions with mnemonic, op code, number of bytes of memory required, number of CPU states, and time in microseconds with a 500KHz clock.

Mnem.	Code	Description*	Mnem.	Code	Description
IN0	101	Interrupt control	OUT20	141	
IN1	103	Keyboard	OUT21	143	
IN2	105	Tape data in/out	OUT22	145	
IN3	107	Tape control in/out,	OUT23	147	
IN4	111		OUT24	151	
IN5	113		OUT25	153	
IN6	115		OUT26	155	
IN7	117		OUT27	157	
OUT10	121		OUT30	161	Right binary display
OUT11	123		OUT31	163	Left binary display
OUT12	125		OUT32	165	
OUT13	127		OUT33	167	
OUT14	131		OUT34	171	
OUT15	133		OUT35	173	
OUT16	135		OUT36	175	
OUT17	137		OUT37	177	

*The description column contains information on the current assignments mentioned in the course of this series. If you make your own hardware, this sheet can be used as a central reference point by filling in your own definitions as notes in the description column. Future articles will add further definitions oriented to ECS series software and hardware designs.

M	OP	L	S	T
ACA	210	1	5	20
ACB	211	1	5	20
ACC	212	1	5	20
ACD	213	1	5	20
ACE	214	1	5	20
ACH	215	1	5	20
ACI	014	2	32	
ACL	216	1	5	20
ACM	217	1	5	32
ADA	200	1	5	20
ADB	201	1	5	20
ADC	202	1	5	20
ADD	203	1	5	20
ADE	204	1	5	20
ADH	205	1	5	20
ADI	004	2	32	
ADL	206	1	5	20
ADM	207	1	5	32
CAL*	106	3	11	44
CFC	102	3	9/11	36/44
CFP	132	3	9/11	36/44
CFS	122	3	9/11	36/44
CFZ	112	3	9/11	36/44
CPA	270	1	5	20
CPB	271	1	5	20
CPC	272	1	5	20
CPD	273	1	5	20
CPE	274	1	5	20
CPH	275	1	5	20
CPI	074	2	32	
CPL	276	1	5	20
CPM	277	1	5	32
CTC	142	3	9/11	36/44
CTP	172	3	9/11	36/44
CTS	162	3	9/11	36/44
CTZ	152	3	9/11	36/44
DCB	011	1	5	20
DCC	021	1	5	20
DGD	031	1	5	20
DCE	041	1	5	20
DCH	051	1	5	20
DCL	061	1	5	20
HLT	000	1	x	x
	001	1	x	x
	377	1	x	x
INB	010	1	5	20
INC	020	1	5	20
IND	030	1	5	20
INE	040	1	5	20
INH	050	1	5	20
INL	060	1	5	20

Input - see separate list...

M	OP	L	S	T
JFC	100	3	9/11	36/44
JFP	130	3	9/11	36/44
JFS	120	3	9/11	36/44
JFZ	110	3	9/11	36/44
JMP	104*	3	11	44
JTC	140	3	9/11	36/44
JTP	170	3	9/11	36/44
JTS	160	3	9/11	36/44
JTZ	150	3	9/11	36/44
LAB	301	1	5	20
LAC	302	1	5	20
LAD	303	1	5	20
LAE	304	1	5	20
LAH	305	1	5	20
LAI	006	2	32	
LAL	306	1	5	20
LAM	307	1	5	32
LBA	310	1	5	20
LBC	312	1	5	20
LBD	313	1	5	20
LBE	314	1	5	20
LBH	315	1	5	20
LBI	016	2	32	
LBL	316	1	5	20
LBM	317	1	5	32
LCA	320	1	5	20
LCB	321	1	5	20
LCD	323	1	5	20
LCE	324	1	5	20
LCH	325	1	5	20
LGI	026	2	32	
LGL	326	1	5	20
LGM	327	1	5	32
LDA	330	1	5	20
LDB	331	1	5	20
LDC	332	1	5	20
LDE	334	1	5	20
LDH	335	1	5	20
LDI	036	2	32	
LDL	336	1	5	20
LDM	337	1	5	32
LEA	340	1	5	20
LEB	341	1	5	20
LEC	342	1	5	20
LED	343	1	5	20
LEH	345	1	5	20
LEI	046	2	32	
LEL	346	1	5	20
LEM	347	1	5	32

Alternatives for conditional instructions: short time if false branch, long time if true branch.

M	OP	L	S	T
LHA	350	1	5	20
LHB	351	1	5	20
LHC	352	1	5	20
LHD	353	1	5	20
LHE	354	1	5	20
LHI	056	2	32	
LHL	356	1	5	20
LHM	357	1	5	32
LLA	360	1	5	20
LLB	361	1	5	20
LLC	362	1	5	20
LLD	363	1	5	20
LLE	364	1	5	20
LLH	365	1	5	20
LLI	066	2	32	
LLM	367	1	5	32
LMA	370	1	7	28
LMB	371	1	7	28
LMC	372	1	7	28
LMD	373	1	7	28
LME	374	1	7	28
LMH	375	1	7	28
LMI	076	2	9	36
LML	376	1	7	28
NDA	240	1	5	20
NDB	241	1	5	20
NDC	242	1	5	20
NDD	243	1	5	20
NDE	244	1	5	20
NDH	245	1	5	20
NDI	044	2	32	
NDL	246	1	5	20
NDM	247	1	5	32
NOP*	300	1	5	20
ORA	260	1	5	20
ORB	261	1	5	20
ORC	262	1	5	20
ORD	263	1	5	20
ORE	264	1	5	20
ORH	265	1	5	20
ORI	064	1	5	20
ORL	266	1	5	20
ORM	267	1	5	32

Output - see separate list...

RAL	022	1	5	20
RAR	032	1	5	20
RET*	007	1	5	20
RFC	003	1	3/5	12/20
RFP	033	1	3/5	12/20
RFS	023	1	3/5	12/20
RFZ	013	1	3/5	12/20

M	OP	L	S	T
RLC	002	1	5	20
RRC	012	1	5	20
RST0	005	1	5	20
RST1	015	1	5	20
RST2	025	1	5	20
RST3	035	1	5	20
RST4	045	1	5	20
RST5	055	1	5	20
RST6	065	1	5	20
RST7	075	1	5	20
RTC	043	1	3/5	12/20
RTP	073	1	3/5	12/20
RTS	063	1	3/5	12/20
RTZ	053	1	3/5	12/20
SBA	230	1	5	20
SBB	231	1	5	20
SBC	232	1	5	20
SBD	233	1	5	20
SBE	234	1	5	20
SBH	235	1	5	20
SBI	034	2	32	
SBL	236	1	5	20
SBM	237	1	5	32
SUA	220	1	5	20
SUB	221	1	5	20
SUC	222	1	5	20
SUD	223	1	5	20
SUE	224	1	5	20
SUH	225	1	5	20
SUI	024	2	32	
SUL	226	1	5	20
SUM!	227	1	5	32
XRA	250	1	5	20
XRB	251	1	5	20
XRC	252	1	5	20
XRD	253	1	5	20
XRE	254	1	5	20
XRH	255	1	5	20
XRI	054	1	5	20
XRL	256	1	5	20
XRM	257	1	5	20

*Instructions marked with asterisk are typical of several alternate op codes, same function.

Arithmetic/Logical Mnemonics:
 AC = add with carry input
 AD = add, no carry input
 SB = subtract, borrow input
 SU = subtract, no borrow input
 ND = logical product (AND)
 OR = logical sum (OR)
 XR = exclusive or (XOR)
 CP = compare

Key: "M"-mnemonic "OP" - op code "L" - length "S" - states "T" - time @500Khz (us)