CA-20

8 Port Input Output

With Real Time Clock

Interface Operation Manual

PRELIMINARY

May 1980

TABLE OF CONTENTS

CA-20
8 Port Input Output
With Real Time Clock
Interface Operation Manual

AN INTRODUCTION TO THE OHIO SCIENTIFIC SIXTEEN PIN I/O BUS

Ohio Scientific is pleased to introduce a unique new product line - The 16 Pin I/O BUS. With this system, it is possible to add up to eight special function boards while occupying only the backplane slot.

This is made possible by a novel BUS extension method which allows decoding, timing and eight bits of data to be carried on standard, inexpensive 16 pin ribbon cables.

Up to eight inexpensive 16 pin cables with standard DIP connectors may be attached to a single CA-20 board which in turn occupies one slot of the standard Challenger backplane. Alternately, one 16 pin I/O BUS cable may be attached to the CA-15 board at the rear of all C4P and C8P products. Note, in the case of the C4P-MF this allows system expansion beyond the normal four slot backplane.

Currently five HEAD END CARDS are available for interconnection to the system via the CA-20 or CA-15 boards.

Computer Interface to Sixteen Pin I/O BUS

The 16 pin I/O BUS may be attached to your computer via two different boards - the CA-15 or the CA-20. The descriptions of these boards are as follows:

CA-15 Board

The CA-15 board is a standard accessory interface installed on the following Ohio Scientific systems: C4P-MF, C4P-DMF, and C8P-DF.

The CA-15 is mounted at the rear of the computer and contains the following interface connections:

Joystick and numeric keypad
Modem and serial printer
Sixteen PIA lines (normally used for the Home Security system - AC-17P)
Sixteen Pin I/O BUS

AN INTRODUCTION TO THE OHIO SCIENTIFIC SIXTEEN PIN I/O BUS

Ohio Scientific is pleased to introduce a unique new product line - The 16 Pin I/O BUS. With this system, it is possible to add up to eight special function boards while occupying only the backplane slot.

This is made possible by a novel BUS extension method which allows decoding, timing and eight bits of data to be carried on standard, inexpensive 16 pin ribbon cables.

Up to eight inexpensive 16 pin cables with standard DIP connectors may be attached to a single CA-20 board which in turn occupies one slot of the standard Challenger backplane. Alternately, one 16 pin I/O BUS cable may be attached to the CA-15 board at the rear of all C4P and C8P products. Note, in the case of the C4P-MF this allows system expansion beyond the normal four slot backplane.

Currently five HEAD END CARDS are available for interconnection to the system via the CA-20 or CA-15 boards.

Computer Interface to Sixteen Pin I/O BUS

The 16 pin I/O BUS may be attached to your computer via two different boards - the CA-15 or the CA-20. The descriptions of these boards are as follows:

CA-15 Board

The CA-15 board is a standard accessory interface installed on the following Ohio Scientific systems: C4P-MF, C4P-DMF, and C8P-DF.

The CA-15 is mounted at the rear of the computer and contains the following interface connections:

Joystick and numeric keypad
Modem and serial printer
Sixteen PIA lines (normally used for the Home Security system - AC-17P)
Sixteen Pin I/O BUS

The features of the clock subsystem are as follows:

Hours, minutes, seconds and 1/10 seconds
Day of week
Day of month
Month of year
Four Year calendar

If you happen to own (or use) a C2 series or C3 series computer, the CA-20 board can actually control the power cycling of the entire computer when equipped with an optional power sequencer package. This means you can preset a time (month, day, hour, etc.) within the clock subsystem and when that preset time agrees with the actual time, A.C. power is applied to the entire computer system through the power sequencer. At a later time, the system's A.C. power may also be removed and the system shut down under software/clock subsystem control.

For applications where the clock subsystem is not required, the CA-20A will perform all the Sixteen Pin I/O BUS functions associated with full-feature CA-20.

HEAD END CARDS

HEAD END CARDS is a general name used to describe any or all of the special function boards which attach to the Ohio Scientific Sixteen Pin I/O BUS. There are currently five such boards and, with the exception of the CA-22, they will only interface with the computer via the Sixteen Pin I/O BUS.

Please note, as detailed earlier, you must use a CA-15 or a CA-20 board at the "computer end" of the Sixteen Pin I/O BUS to complete the interface.

In the following pages a brief product and application description of the currently available HEAD END CARDS will be presented.

Bit Switching and Sensing —— The CA-21

The CA-21 is a 48 line parallel I/O board featuring three 6821 PIAs (peripheral interface adapters) and prototyping/interconnect areas.

The use of PIAs in the design allows for maximum interface versatility as you may configure any one of the 48 I/O lines as either an input or an output. As outputs, each line is capable of driving a minimum of one standard TTL load.

Additional versatility is added because 24 of the lines, when configured as outputs, may simultaneously function as inputs. This feature, although somewhat confusing, is extremely useful for applications such as switch matrix decoding.

Each of the 48 lines is brought out to two foil pads (suitable for wire wrap stakes) as well as a location on one of four 12 pin Molex-type female edge connectors. There are also eight 16 pin DIP socket locations which are intended for use as prototyping areas. Additionally, the 12 PIA "hand-shaking" lines are brought to 12 single foil pads.

The CA-21, with proper buffering, may be used for virtually any computer controlled bit switching or bit sensing application that you can imagine. With a full complement of eight CA-21s interfaced via the CA-20, a total of 384 individually controllable I/O lines are possible!

An interesting application using one CA-21 board would be a complete, if somewhat slow, emulation of the standard Ohio Scientific BUS.

A more standard application might be augmenting the standard Home Security System (AC-17P) with "hard-wired" sensors.

One type of sensor you could easily add is a standard window "perimeter detector". This could be done with commercially available adhesive foil tape. You could then detect a break-in (through a broken window) by sensing a break in the foil tape.

Another useful application you could set up in concert with the AC-12P wireless A.C. Remote Control, might be sensing when a room is entered. You could accomplish this with pressure-switch door mats or door switches. When room entry is detected, the lights could be turned on or, turned off on exit.

If you are designing any sort of dedicated control system, the CA-21 is an ideal choice. You can easily sense many types of input (pressure transducers, flow sensors, switches, etc.) while controlling outputs from a simple single LED display to a network of solid state relays controlling A.C. power.

EPROM Programmer —— The CA-23

The CA-23 is an EPROM programmer designed for use with the growing families of 5 volt only EPROMS. With the CA-23 you can program and verify all 1K through 8K byte EPROMS of this type. Note these parts are often identified as 8K - 64K bit EPROMS.

The CA-23 can program (or verify) data in two basic modes - EPROM to/from EPROM or EPROM to/from computer RAM memory.

Additionally, EPROM data may be read directly into the computer's RAM memory.

There are four LED indicators on the CA-23. The first is "SOCKET UNSAFE". This means that a programming voltage is present at the socket and if you insert or remove an EPROM it is likely to be damaged.

The second indicator is "PROGRAMMING". This means that your EPROM is currently being programmed.

The third indicator is "ERROR". This means that somewhere along the line your programming attempt was unsuccessful.

The final indicator is "PROGRAM COMPLETE". This means that your program and verification was successful.

The most intriguing application for this product is the creation of "custom" parts for your computer or peripherals. This could range from a new system monitor to a new high level language. It could even include a new character generator for your CRT or printer. Note, however, tinkering around with the internals of computers and peripherals requires a fairly high degree of technical expertise. Also, most manufacturer's warranties are voided by these types of modifications.

Several OEM (original equipment manufacture) and Research/ Development applications will be immediately obvious to those of you involved in that work.

The CA-23, as previously mentioned, is designed for use with 1K through 8K byte EPROMS. These parts come in various package styles and have various product names. For example, Intel's 2K x 8 part is the 2716, Texas Instruments' part is known as the 2516.

The CA-23 has both 24 pin and 28 pin zero insertion force sockets for reading, programming and verifying the EPROMS.

Prototyping —— The CA-24

The CA-24 is a solderless bread-board designed for prototyping, experimental and educational applications.

The bread-boarding is made up of seven solderless plug-strips of the type manufactured by AP Products. Two of the plug-strips contain a connection matrix of 5 by 54 connections and are used as signal distribution points. Another pair of 96 location plug-strips are for powering the bread-board area. The actual experimenter area is comprised of three plug-strips, each with a 10 by 64 location connection matrix. Additionally, sixteen LED indicators and sixteen DIP switch positions are provided for signal observation and control functions.

Board I/O is via TTL latches and bi-directional PIA ports as well as direct (buffered) data, signal and control lines from the computer BUS. This method allows you to directly interconnect devices such as 6850 ACIAs in addition to doing more "isolated" and/or independent circuits.

The CA-24 also contains a "clock" generator which is continuously variable from approximately 25,000 Hz. through 70,000 Hz. You may also connect the clock to an on-board 16 stage divider chain. This allows division of the fundamental frequency by as little as 2^1 (2) to as much as 2^{16} (65,536).

The applications for the CA-24 are primarily prototyping and experimenting. Parts may be inserted and removed from the terminal strip blocks over and over. Interconnection of parts is accomplished simply with solid, narrow gauge wire jumpers. Errors in design or connection are extremely easy to correct.

The CA-24 lends itself very well to structured experiments that are common in the educational environment. It is an ideal tool to aid in the teaching of computer and computer interface fundamentals.

<u>Accessory Interface —— The CA-25</u>

The CA-25 is designed to implement some of the functions normally associated with the CA-15 interface board.

It allows you to directly connect the Home Security System (AC-17P) and/or the Wireless A.C. Remote Control System (AC-12P) to C2 and C3 series computers. Additionally, if you own an older Ohio Scientific computer, you can now easily connect these systems to it.

An extremely useful application of the CA-25 is associated with small business systems. Using the CA-25 with the Home Security System, and perhaps a CA-15V (Universal Telephone Interface with speech synthesizer output), the computer could do payroll, inventory, etc. by day and "guard" the shop by night.

<u>Analog I/O —— The CA-22</u>

The CA-22 is a high speed analog I/O module. Although the CA-22 is classified as a HEAD END CARD, it differs from the rest of the family in that it may also be plugged directly into the computer's standard internal BUS. This allows for maximum flexibility in the use of the CA-22.

The analog input section of the CA-22 consists of a 16 channel analog multiplexer. This means that you may connect up to 16 separate signals directly to the CA-22. Also included is a sample and hold circuit followed by the analog to digital converter circuitry.

The A to D converter is capable of either 8 bit or 12 bit operation. You may select these options under software control.

The accuracy of the converter is plus or minus one in the least significant bit. The stability of the circuit is rated at one millivolt drift per degree Centigrade.

The A to D conversion is extremely fast. It is capable of digitizing up to 66,000 samples per second in the 8 bit conversion mode and 28,000 samples per second in the 12 bit mode. Shannon Sampling Theory states that signals should be sampled at twice the highest frequency present. Therefore, it is possible for you to convert signals with a frequency greater than 30K Hz. Clearly, high fidelity audio is well within the spectrum of the CA-22.

The multiplexer has very high impedance inputs and is capable of accepting inputs in the range of -10 volts through +10 volts. The input is jumper selectable for other settings including a single sided range of 0 through +10 volts.

Due to the indeterminable nature of the actual inputs that you may actually apply to the CA-22, only the multiplexer inputs are brought out. However, a quad op-amp is laid out in foil which you may populate in several different modes to handle some of the more "common" input configurations.

The analog output section of the CA-22 consists of two identical high speed digital to analog converters. Each DAC can convert either 8 bits or 12 bits of data. Data input to the DACs is latched in such a manner that, when in the 8 bit conversion mode, the other four (of the total of twelve) bits are continuously output at a predefined value. You may, of course, define that value under software control.

The output of each DAC is buffered with a high speed op-amp capable of changing output voltage at the rate of 20 volts per microsecond. The standard configuration of each output is bi-polar with a voltage swing of -10 volts through +10 volts. This is jumper selectable to allow a uni-polar output of 0 through +10 volts.

Some additional I/O capacity is provided on the CA-22. There are three TTL level inputs and six open collector logic outputs. These are strappable to be either standard TTL level outputs or high-voltage outputs.

You can use the CA-22 for a multitude of analog sensing and/or analog controlling applications.

Using the proper transducers and the 16 input channels, you can monitor the temperature in several zones of a home or office. By extending this system with a CA-21, you could maintain precise temperatures by switching the proper controls on and off.

Another interesting, if somewhat obvious application, is in audio processing. Reverberation, phase shifting and echoing are just a few of the uses you could implement.

If you used blocks of RAM for data storage, other applications such as frequency doubling, etc., could be experimented with.

If you apply more sophisticated software techniques, such as a fast Fourier transform, on stored input data, very elaborate signal processing becomes realizable. Projects such as audio spectrum analyzers and speech recognition experiments are certainly practical. Note, in these types of applications you are likely to find some signal pre-processing in hardware is certainly beneficial - if not totally necessary.

If you employ both DAC outputs and the on-board unblanking circuit, X-Y oscilloscope plotting is an interesting application. By using these techniques and one or more of the analog inputs, you can construct a digital storage scope. Note, both of these applications require that you have access to an oscilloscope capable of X-Y input as well as blanking.

Summary

With the introduction of the 16 pin I/O BUS, Ohio Scientific has opened a new world of interfacing capabilities for both the large and the small computer user.

Systems ranging from totally automated sampling and control stations to complete R/D setups to educational lab stations are now available to you via standard building blocks and standard computer systems.

For pricing and availability, contact your nearest Ohio Scientific dealer.

Purpose and Scope

The CA-20 is designed to expand the I/O capabilities of the OSI Challenger Series. Each CA-20A is capable of supporting up to eight "HEAD END CARDS" away from the computer card cage while requiring only one card slot of the motherboard for the interface. The CA-20, in addition to the features of the CA-20A, also includes a time of day clock and real time clock with interrupts or polling capability. The clock is supported by battery power which is automatically recharged whenever the computer is turned on. The CA-20 is installed inside the computer mainframe and the accessory boards (HEAD END CARDS) are outside of it, virtually increasing I/O capability to any desired amount. Several CA-20 boards could be installed in one computer by readdressing the CA-20 with jumper connections provided on the board.

CA-20, CA-21, CA-22 Board Software

OS-65D Process Control Basic has six reserved words interfacing with the CA-20, CA-21 and CA-22 boards. These are the DGIN, DGOUT, ANIN, ANOUT, TIME and DATE statements. This software assumes that a CA-22 must be located at PORT Ø of a CA-20 or J2 of a C4P or C8P, or inserted directly into the rack at address $C7Ø4_{16}$, if ANIN and ANOUT are to work. For TIME and DATE, to use the CA-20 programmable clock, it must be located at $C78Ø_{16}$ (as shipped). Otherwise, a polled clock and calendar will be maintained by monitoring the one-second ticks of the 470 floppy controller board.

DGIN Base, Bit, Variable

DGOUT Base, Bit, Value

DGIN reads the CA-21 board. A PIA bit and register are selected and read and a zero or one stored in variable.

DGOUT writes the CA-21 board. A PIA bit and register are selected, value is evaluated to be zero or non-zero, and a zero or one written, sending the data line low or high, respectively.

Base is an expression yielding the address of the desired PIA data register.

Bit is an expression yielding a byte value selecting a data line (bit Ø-7 of a data register). If the bit selected is greater than 7, higher PIA registers are addressed

(address=base+[(bit/8)*2]),

and the line selected is mod_8 (bit). Values larger than a byte get a function call error. Base and bit are similar in behavior to the address and byte of the POKE statement.

Variable is any legal BASIC integer or floating variable.

Value is any legal BASIC integer or floating expression.

A syntax error is the normal result of misuse, but bad addresses can cause crashes when the attempted initialization of a PIA for reading or writing is done over operating system addresses or other forbidden locations.

ANIN Channel, Variable

ANIN reads the CA-22 board for analog-to-digital data.

Channel is an expression with a value from Ø-15, which is used to select an input line to be read. Variable is any legal BASIC floating or integer variable, which will store a value from Ø to 255 (integers) or 255.9375 (floating). A full scale is used. (For the -1Ø volts to +1Ø volts setting, this means that -1Ø volts would yield a Ø, Ø volts a 128, and +1Ø volts a 255 or 255,9375.)

Twelve bit mode is always selected, but use of eight bits merely results in a fraction of zero.

ANOUT Dac, Value

ANOUT uses the CA-22 board to output digital-to-analog data.

Dac is an expression yielding a value of zero or one, selecting digital-to-analog converters one or two, respectively.

Value is any legal BASIC integer or floating expression with a result greater than or equal to Ø and less than 256. Twelve bits will always be output, but the four least significant bits for an integer value will be zero.

```
TIME=Hours{, Minutes{, Seconds}}
TIME Hours, Minutes, Seconds
DATE=Month, Day, Year
DATE Month, Day, Year
```

TIME= sets the system time-of-day clock. The CA-20 is used, if possible. Otherwise, a polled clock is maintained using the one-second ticks from the 470 floppy controller. The choice is made after some simple tests for system configuration are made during the bootstrap procedure. Only the first parameter is mandatory. All parameters may be any legal BASIC integer or floating expression with results within parameter ranges.

TIME stores the system time-of-day values into the floating or integer variables provided by the user as parameters. All three are mandatory.

DATE= sets the system calendar using hardware as the TIME= statement does. All three parameters are required and may be any legal BASIC expressions.

DATE stores the system calendar values into the floating or integer variables listed as parameters.

Hours ranges from Ø to 23, as the system time uses a 24-hour nomenclature.

Minutes and seconds range from Ø to 59.

Month ranges from 1 to 12.

Day ranges from 1 to 31 and is checked against year and month for a legal Gregorian date.

Year may be two or four digits. Two digit values are added to 19ØØ (all legal Gregorian dates are four digit).

The following discussion of available machine code subroutines is merely to describe to the potential user their availability and use. The subroutine addresses may change after this paper is published. Refer to the documentation supplied with (Process Control Basic) for the addresses to call in your version.

The process control extensions to BASIC are always resident and may be accessed from assembly programs whether BASIC is loaded or not. When these routines error out, however, they will attempt to JMP to BASIC error reporting routines.

DGIN subroutines:

DIASM1 - set selected bits of a PIA data register to input mode. Store the data register address at 19_{16} (low) - $1A_{16}$ (high). Store a byte in X with zeros in the bits to become input lines and ones in the lines to be untouched. JSR to 3663_{16}.

DIASM2 - read a PIA data register. Store the data register address in 19_{16} (low) - $1A_{16}$ (high). JSR to $367Ø_{16}$. The read byte will be returned in A.

DRSTOR - restore PIA to previous state. JSR to $36FØ_{16}$.

DGOUT subroutines:

DOASM1 - set selected bits of a PIA data register to output mode. Store the data register address at 19_{16} (low) - $1A_{16}$ (high).

Store a byte in X with zeros in the bits to become output lines and ones in the lines to be untouched. JSR to $369D_{16}$.

DORITE - store a data byte into the data register. Store the data register address at 19_{16} (low) - $1A_{16}$ (high). Set Y to zero and put the data byte in A. If used with DOASM1, the control register will be restored. JSR to 3695_{16}. If control register restoration is undesirable, set up as for DORITE but JSR to $36A4_{16}$.

ANIN subroutines:

AIASM1 - select 12 bit mode and set a channel for input. Load the channel number (Ø-F_{16}), into register X and JSR to 3718_{16}.

AIASM2 - read the currently selected channel until the valid data flag indicates the read is good. If the flag fails to go valid in 255Ø clock cycles, this routine will JMP to $44E_{16}$, the BASIC error reporter. JSR to $372B_{16}$. The eight most significant bits will be returned at AF_{16}, the four least significant bits will be returned in bits Ø-3 of $BØ_{16}$.

ANOUT subroutines:

Store the 8 MS bits at AF_{16}, the 4 LS bits, if desired, in bits Ø-3 of $BØ_{16}$. Set Y to Ø or 2, selecting DAC 1 or 2 and JSR to 3777_{16} for 12 bits, or load A with Ø and JSR to $377D_{16}$, for 8 bits.

TIME subroutines:

When the polled clock is in use, the hours, minutes, and seconds values are stored at $37C2_{16}$ - $37C4_{16}$, respectively. To maintain the polled clock, JSR to $37C5_{16}$ at sub-second intervals. To tell whether the polled or programmable clock is in use, location $37FD_{16}$ is Ø if polled, FF_{16}, if programmable.

To set the programmable clock, load 3961_{16} - 3963_{16} with hours, minutes, and seconds, respectively. JSR to $386F_{16}$ to set seconds, minutes, and hours; $387C_{16}$ to set minutes and hours; 3887_{16} to set hours. To write any clock register, store the register number and value at $39A6_{16}$ - $39A7_{16}$, respectively, in BCD form and JSR to $399A_{16}$.

To read the programmable clock, JSR to $383D_{16}$ and hours, minutes, and seconds will be stored at 3961_{16} - 3963_{16}, respectively. To read hours, minutes, or seconds individually, JSR to $38B8_{16}$, $38A9_{16}$, or $389A_{16}$, respectively. To read any clock register, store the register number in BCD at $39A6_{16}$, and JSR to 3984_{16}. The value will be returned at $39A7_{16}$, in BCD.

Software

Address Determination

Each "HEAD END CARD" can have up to 12 distinct sub-port addresses which are required to accommodate the various functions of each board. An example of this is the CA-21 which has three PIA's (peripheral interface adapter). Each PIA has four distinct register locations internal to it which must be accessed independently. Therefore, for correct processor communication with this accessory board, 12 independent and distinct 8 bit register locations must be accessed (read or write). This is accomplished by the above mentioned subset of port addresses. To rephrase that, each of the eight ports has 12 separate sub-port addresses that will enable that port (read or write) and each sub-port address will enable an individual sub-port on the "HEAD END CARD" if so equipped.

The address subset (RANGE) of each port is determined in the following manner. First the board select address range of the CA-20 is found by determining the hardware configuration of the address decoders. The CA-20 as shipped from the factory is $C7XX (hex). Conversion to decimal will result in a board select range of 50944 to 51199 decimal. If the board select address has been modified by the user then these figures will have to be changed as required. Each individual base port address can now be found by BASE ADD. = BS+(PORT #X16). Where BS is the board select address (50944) as shipped from the factory and PORT # is the desired PORT from 0 to 7. Assume you wish to find the base address of PORT #5. Using the above equation, it is found that the base address is 50944 + (5X16) = 51024. To find the entire subset of the PORT add 15 to the base address. For PORT #5 the range is 51024 to 51039. Notice that the entire port subset actually includes 16 distinct addresses but only 12 will ever be used. The first four of each port (for PORT #5 51024 to 51027) should not be used and the upper 12 are the actual subset. If any of the first four of the port subset are accessed the "HEAD END CARD" will not acknowledge and no data will be transferred in either direction (read or write). The only thing that would happen if any of these are accessed is wasted machine cycles.

To find the sub-port address for a given location or register on a "HEAD END CARD" connected to any port, the sub-port address number should be added to the port base address. For PORT #5 this would be 51024 + sub-port number. The sub-port number is found by converting the least significant hex-digit of the hex address found in the "HEAD END CARD" manual to decimal. For example: If the "HEAD END CARD" register address desired was specified as $C70A then the (A) would be converted to decimal (10). If this card is connected to PORT #5 of the CA-20 then this sub-port address would be 51024 + 10 = 51034. In most "HEAD END CARD manuals the sub-port number will be identified in the software section. Under no circumstances should the sub-port number be less than 4 or greater than 15.

The previous method will work with all "HEAD END CARDS" installed in any one of the eight ports and may be useful in generating efficient code for multiple "HEAD END CARD" installation.

It should be noted that this is not a necessary function at run time and should be avoided especially if a low number of cards are being interfaced. In this case the addresses should be precomputed and inserted directly into the code (variable would be faster) to increase processing speed. It should also be noted that if a "HEAD END CARD" is installed in PORT Ø, the "ADDRESS MAP" in the "HEAD END CARD" manual may be used directly since PORT Ø has the same port subset addresses as the original OSI accessory bus. The machine code programmer should notice that each port is identical in structure of addressing and each is 16($F) from the last/next.

Basic PROG #8 could be used to read the sub-ports of a PIA on the CA-21 "HEAD END CARD" with data registers at sub-ports 4 and 6. This program assumes that the registers have already been defined as inputs and is simply a demonstration of one possible method of generating the sub-port addresses required. If the CA-21 were moved to another port changing the number (5) in line 60 would redefine the address of that card. If the CA-21 were installed at PORT Ø this number would be changed to (DEV=Ø).

Basic PROG #9 builds upon this addressing scheme and does two operations. First it initializes all six data registers as inputs from an unknown state, then reports the results of reading the six registers in binary. This program can be directed to any port by changing the 5 in line 140 to the desired PORT number. It is beyond the scope of this paper to cover techniques used in PIA programming. PROG #9 is merely a demonstration of a possible addressing scheme that could be used to generate efficient code for multiple sub-port interfacing.

In PROG #A we will cover a more difficult task. Let's assume that the CA-20 has been populated with three CA-21 "HEAD END CARD" interfaces and for some reason are randomly connected to PORTS 2, 4 and 7. PROG #A will perform the same task as PROG #9 but will be interfacing to three PORTS and 18 input data registers. The program could be easily expanded to handle up to eight CA-21 cards by modifying lines 130 and 140. Instead of simply displaying data, the data could be stored for later reference or acted upon immediately.

```
10 GOTO40                  PROG #8
15 :
20 :        ASSUMES CA-21 ALREADY INITIALIZED AS-INPUTS
30 :        READ TWO SUB-PORTS 4 AND 6 AND DISPLAY BINARY
35 :
40 :        BASE=50944                 :REM CA-20 BASE SELECT ADD.
60 :        DEV=5                      :REM DEVICE PORT NUMBER
80 :
90 :PRINTTAB(37);"76543210"            :REM PRINT DISPLAY FORMAT
95 :
100 FOR SUB = 4 TO 6 STEP2             :REM SUB-PORT NUMBER 4 AND 6
120 :   ADD=BASE+SUB+(DEV*16)          :REM FIND SUB-PORT ADDRESS
140 :   DTA=PEEK(ADD)                  :REM READ SUB-PORT DATA
160 :   PRINT"DATA READ AT PORT#";
170 :   PRINTDEV"SUB-PORT#"SUB"WAS ";
175 :
180 :      FOR DIGIT = 7 TO 0STEP-1:REM PRINT DATA IN BINARY
190 :         HEX=DTA AND 2^DIGIT
200 :         IFHEX<>0 THEN PRINT"1";
210 :         IF HEX=0 THEN PRINT"0";
220 :      NEXT DIGIT
225 :
230 :      PRINT
235 :
240 NEXT SUB
250 :
260 END:OF:PROGRAM
```

```
10 GOTO 120           PROG #9
20 :
40 :      INITIALIZE ALL INPUTS ON CA-21 CARD AS-INPUTS
60 :      READ ALL INPUT REGISTERS AND DISPLAY
80 :      AS BINARY. STOP
100 :
120 BASE=50944                  :REM BASE ADDRESS CA-20
140 DEV=5                       :REM PORT LOCATION
160 :
185 ADD=BASE+(DEV*16)           :REM CALCULATE SUB-PORT BASE
190 :
200 REM (INITIALIZE) ALL REGISTERS AS-INPUTS
220 FOR SUB=15 TO 4 STEP -1     :REM SUB-PORT NUMBER
240 : POKE ADD+SUB,0            :REM STASH 0 IN SUB-PORTS
260 NEXT                        :REM CONTINUE TILL ALL DONE
270 :
280 FOR SUB=5TO15STEP2          :REM 5,7,9,11,13,15 CONT.REG.
300 : POKE ADD+SUB,4            :REM SET DDR/DTA TO DTA
320 NEXT                        :REM (INITIALIZE) COMPLETE
340 :
360 REM READ ALL INPUT REGISTERS AND DISPLAY AS BINARY
370 PRINTTAB(40);"76543210"    :REM DISPLAY FORMAT
375 :
380 FOR SUB = 4 TO 14 STEP2     :REM 4,6,8,10,12,14 DTA REG.
400 : DTA=PEEK(ADD+SUB)         :REM GET DATA FROM REGISTER
410 : PRINT"DATA READ AT PORT#";
420 : PRINTDEV"SUB-PORT#"SUB;TAB(34);"WAS";
440 : PRINTTAB(40);
450 :
460 :   FOR DIGIT = 7 TO 0 STEP-1
470 :    HEX = DTA AND 2^DIGIT
480 :    IF HEX<>0 THEN PRINT"1";
500 :    IF HEX=0 THEN PRINT"0";
520 :   NEXT DIGIT
525 :
530 : PRINT
535 :
550 NEXT SUB
570 :
580 END:OF:PROGRAM
```

```
10 GOTO 120            PROG #A
20 :
40 :      INITIALIZE ALL INPUTS ON CA-21 CARDS AS-INPUTS
60 :      READ ALL INPUT REGISTERS AND DISPLAY
80 :      AS BINARY. STOP
100 :
120 BASE=50944                  :REM BASE ADDRESS CA-20
130 CARDS=3                     :REM NUMBER OF CA-21 CARDS
140 DATA 2,4,7                  :REM PORT LOCATION OF CARDS
160 :
170 REM DO FOR AS MANY CARDS AS THERE ARE
180 FOR JOB=1TOCARDS:READ DEV    :REM GET PORT NUMBER
185 : ADD=BASE+(DEV*16)          :REM CALCULATE SUB-PORT BASE
190 :
200 : REM INITIALIZE ALL REGISTERS AS-INPUTS
220 : FOR SUB=15 TO 4 STEP -1    :REM SUB-PORT NUMBER
240 :   POKE ADD+SUB,0           :REM STASH 0 IN SUB-PORTS
260 : NEXT                       :REM CONTINUE TILL ALL DONE
270 :
280 : FOR SUB=5TO15STEP2         :REM 5,7,9,11,13,15 CONT.REG.
300 :   POKE ADD+SUB,4           :REM SET DDR/DTA TO DTA
320 : NEXT                       :REM INITIALIZE COMPLETE
340 :
360 : REM READ ALL INPUT REGISTERS AND DISPLAY AS BINARY
370 : PRINTTAB(40);"76543210"  :REM DISPLAY FORMAT
375 :
380 : FOR SUB=4TO14STEP2         :REM 4,6,8,10,12,14 DTA REG.
400 :   DTA=PEEK(ADD+SUB)        :REM GET DATA FROM REGISTER
410 :   PRINT"DATA READ AT PORT#";
420 :   PRINTDEV"SUB-PORT#"SUB;TAB(34);"WAS";
440 :   PRINTTAB(40);
450 :
460 :     FOR DIGIT = 7 TO 0 STEP-1
470 :      HEX = DTA AND 2^DIGIT
480 :      IF HEX<>0 THEN PRINT"1";
500 :      IF HEX=0 THEN PRINT"0";
520 :     NEXT DIGIT
525 :
530 :   PRINT
535 :
540 : NEXT SUB
550 :
560 NEXT JOB
570 :
580 END:OF:PROGRAM
```

TABLE I

CA-20 PORT ASSIGNMENTS

HEX	DECIMAL	PORT #	SUB-PORT #	HEX	DECIMAL	PORT #	SUB-PORT #
C704	50948	0	4	C744	51012	4	4
C705	50949	0	5	C745	51013	4	5
C706	50950	0	6	C746	51014	4	6
C707	50951	0	7	C747	51015	4	7
C708	50952	0	8	C748	51016	4	8
C709	50953	0	9	C749	51017	4	9
C70A	50954	0	10	C74A	51018	4	10
C70B	50955	0	11	C74B	51019	4	11
C70C	50956	0	12	C74C	51020	4	12
C70D	50957	0	13	C74D	51021	4	13
C70E	50958	0	14	C74E	51022	4	14
C70F	50959	0	15	C74F	51023	4	15
C714	50964	1	4	C754	51028	5	4
C715	50965	1	5	C755	51029	5	5
C716	50966	1	6	C756	51030	5	6
C717	50967	1	7	C757	51031	5	7
C718	50968	1	8	C758	51032	5	8
C719	50969	1	9	C759	51033	5	9
C71A	50970	1	10	C75A	51034	5	10
C71B	50971	1	11	C75B	51035	5	11
C71C	50972	1	12	C75C	51036	5	12
C71D	50973	1	13	C75D	51037	5	13
C71E	50974	1	14	C75E	51038	5	14
C71F	50975	1	15	C75F	51039	5	15
C724	50980	2	4	C764	51044	6	4
C725	50981	2	5	C765	51045	6	5
C726	50982	2	6	C766	51046	6	6
C727	50983	2	7	C767	51047	6	7
C728	50984	2	8	C768	51048	6	8
C729	50985	2	9	C769	51049	6	9
C72A	50986	2	10	C76A	51050	6	10
C72B	50987	2	11	C76B	51051	6	11
C72C	50988	2	12	C76C	51052	6	12
C72D	50989	2	13	C76D	51053	6	13
C72E	50990	2	14	C76E	51054	6	14
C72F	50991	2	15	C76F	51055	6	15
C734	50996	3	4	C774	51060	7	4
C735	50997	3	5	C775	51061	7	5
C736	50998	3	6	C776	51062	7	6
C737	50999	3	7	C777	51063	7	7
C738	51000	3	8	C778	51064	7	8
C739	51001	3	9	C779	51065	7	9
C73A	51002	3	10	C77A	51066	7	10
C73B	51003	3	11	C77B	51067	7	11
C73C	51004	3	12	C77C	51068	7	12
C73D	51005	3	13	C77D	51069	7	13
C73E	51006	3	14	C77E	51070	7	14
C73F	51007	3	15	C77F	51071	7	15

Clock Software

Using Standard OS-65D

Table IV lists the maximum used BCD code of each four bit byte (two per word) when reading or writing to the clock. If either of the bytes in the eight bit counter words do not legally reach four bit lengths (e.g., day of the week uses only the three least significiant bits), the unused bits will be unrecognized during a write and held at zero during a read. If any illegal data is entered into the counters during a write cycle, it may take up to four clocks (for months in the case of the month counter) to restore legal BCD data to the counter. The latches will read and write all four bits per byte. Each of the counter and latch words can be reset with the appropriate address and data inputs. The counter reset is a write function. The latches can be programmed to compare with the counters at all times by writing one's (1) into the two most significant bits of each latch, thus, establishing a don't care state in the latch. The don't care state is programmable on the byte level, i.e., tens of hours can contain a don't care state, yet unit hours can contain a valid code necessary for a comparison.

Table II lists the address codes of the clock and their associated functions. These addresses, as all signals to the clock, are implemented in hardware by a PIA, which must be initialized before "talking" to the clock. The initialization procedure for the address lines is

```
POKE 51Ø77,Ø:POKE 51Ø76,31:POKE 51Ø77,4
```

for write set up and

```
POKE 51Ø77,58:POKE 51Ø76,31:POKE 51Ø77,62
```

for a read set up. The address for a clock access cycle can now be written to 51Ø76. Selection of the hour counter would require POKE 51Ø76,4. The address must be written before reading or writing from the clock. Refer to PROG #1, lines 210 and 250 and PROG #2, lines 280, 290, 330.

Writing to the Clock

The PIA interface to the clock must be initialized for writing before attempting to write to the clock. The procedure after setting up the PIA address format is

```
POKE 51Ø79,34:POKE 51Ø78,255:POKE 51Ø79,38.
```

Refer to PROG #1, lines 250 and 310. The address of the desired clock register must now be written by POKE 51Ø76,ADD (PROG #1, line 340). The write data and WRITE command are then performed by POKE 51Ø78,DTA (PROG #1, line 340). Line 350 checks to insure that a complete write cycle has taken place. Line 360 resets CRB-7 which was used internally to the PIA to reset the WRITE command.

When writing to the counters, the data should be checked to insure that an illegal code will not be written. See Table IV. Refer to PROG #1. In line 70, the maximum decimal code for each counter is provided to the variable (CHK) in line 120. Before writing to the counters or latches, the decimal code must be converted to BCD. See PROG #1, lines 370 through 400.

```
10 REM                PROG #1
20 REM PROGRAM TO WRITE TO TIME OF DAY CLOCK
30 REM THIS PROGRAM EXITS WITH THE CLOCK IN WRITE MODE
40 REM WRITE THRU PIA
50 DA=51076:CA=DA+1:REM A DATA/A CONT
60 DB=CA+1:CB=DB+1:REM B DATA/B CONT
70 DATAMONTH,12,DAY OF MONTH,31,DAY OF WEEK,7,HOUR,23,MINUTE,59
80 ACODE=3:MNTH=7:REM MINUTES TO MONTHS
90 GOSUB 200:REM INITIALIZE FOR WRITE
100 ADD=17:GOSUB320:REM MASK ALL INTERRUPTS
110 FORADD=MNTHTOACODESTEP-1:READA$:READ CHK:PRINTA$;:INPUTDTA
120 IF DTA>CHK THEN RUN
130 GOSUB370:GOSUB320:NEXTADD:REM CONV 2-BCD,WRITE
140 ADD=2:DTA=10:GOSUB 320:REM SET SECONDS UNDER 40
150 REM GO ON INPUT
160 ADD=21:DTA=0:REM SET UP FOR GO
170 POKE2888,0:INPUT"HIT RETURN WHEN READY";A$:POKE2888,27
180 GOSUB 320:REM GO COMMAND
190 RUN"PROG2
195 :
200 REM (SUB) INIT PIA/B AND PIA/A AS OUT
210 POKECA,0:POKEDA,31:REM PIA/A ALL OUTS
220 REM CA1-INPUT,CA2=INPUT
230 REM CRA-7 GOES HIGH ON TRAILING EDGE OF INT.REQ
240 REM CA2 HIGH PIA PULLUP USED TO PULL READ HIGH
250 POKECA,4:POKECB,34:POKEDB,255:REM PIA/B ALL OUTS
260 REM CB1 AND CB2=OUTPUT
270 REM CRA-7 GOES HIGH ON ^ EDGE OF READY
280 REM CB2 GOES HIGH ON CRA-7 SET
290 REM CRA-7 RESET BY READING DB
300 REM CB2 GOES LOW WHEN WRITE TO PB0-7
310 POKECB,38:RETURN
315 :
320 REM (SUB) WRITE TO CLOCK
330 REM PASS ADDRESS IN ADD,DATA IN DTA
340 POKE DA,ADD:POKE DB,DTA
350 D=PEEK(CB):D=DAND128:IFD=0 THEN 350
360 D=PEEK(DB):RETURN:REM RESET CRB-7 DATA USELESS
365 :
370 REM (SUB) CONVERT DTA FROM DECIMAL TO 2-BCD
380 IF DTA<10 THEN RETURN
390 TEMP=INT(DTA/10):LOW=DTA-(TEMP*10):TEMP=TEMP*16
400 DTA=TEMP+LOW:RETURN
```

Reading from the Clock

The PIA interface to the clock must be initialized for reading before attempting to read the clock. The procedure after setting up the PIA address format is

```
POKE 51Ø79,58:POKE 51Ø78,Ø:POKE 51Ø79,62.
```

Refer to PROG #2, lines 330 and 380. The address of the desired clock register must now be written by POKE 51Ø76,ADD. See PROG #2, line 150. The READ command is then passed to the clock by POKE 51Ø77,54. See PROG #2, lines 380 through 400. Line 390 waits for valid data. Line 400 gets the read data PEEK (51Ø78) and removes the READ command POKE 51Ø77,62. The data must now be converted from BCD to decimal. See PROG #2, lines 170 through 180.

```
10 REM              PROG #2
20 REM PROGRAM TO READ TIME OF DAY CLOCK
30 FORI=1TO30:PRINT:NEXT
50 REM READ THRU PIA
60 DA=51076:CA=DA+1:REM A DATA/A CONT
70 DB=CA+1:CB=DB+1: REM B DATA/B CONT
80 ACODE=2:MNTH=7:DIM C$(12):REM STOP AT MONTH
90 FORI=1TO12:READC$(I):NEXTI:FORI=1TO7:READD$(I):NEXTI
100 DATA JANUARY,FEBUARY,MARCH,APRIL,MAY,JUNE,JULY,AUGUST
110 DATA SEPTEMBER,OCTOBER,NOVEMBER,DECEMBER
120 DATA SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY
130 GOSUB 270:REM INITIALIZE FOR READ
135 :
140 FOR ADD=MNTHTOACODESTEP-1
150 POKE DA,ADD:REM SET UP CLOCK ADDRESS
160 GOSUB 380:REM READ CLOCK
170 MS=DTAAND240:MS=(MS/16)*10:LS=DTAAND15
180 TIME=MS+LS:REM CONVERT DECIMAL BACK TO 2-BCD
190 T(ADD)=TIME:NEXTADD
200 PRINTCHR$(13);D$(T(5));" ";C$(T(7));T(6);"  ";
210 IF T(4)=>12 THEN PM=1
220 IF T(4)=>13 THEN T(4)=T(4)-12
225 IF T(4)=0 THEN T(4)=12
230 PRINTT(4);":"T(3)":"T(2);"  ";
240 IFPM=1THENPRINT"PM   ";:GOTO260
250 PRINT"AM   ";
260 PM=0:GOTO140:REM GO DO IT AGAIN
265 :
270 REM (SUB) INIT PIA.B AS IN:PIA/A AS OUT
280 POKE CA,58:REM 00111010
290 POKE DA,31:REM PIA/A ALL OUTPUTS
300 REM CA1=INPUT,CA2=OUTPUT
310 REM CRA-7 GOES HIGH ON TRAILING EDGE OF INT.REQ
320 REM CA2 FOLLOWS CRA-3
330 POKECA,62:POKECB,58:POKEDB,0:REM PIA/B ALL INPUTS
340 REM CB1=INPUT,CB2=OUTPUT
350 REM CRB-7 GOES HIGH ON ^ EDGE OF READY
360 REM CB2 FOLLOWS CRA-3 & MUST BE HIGH F.READ
370 POKECB,62:RETURN
375 :
380 POKE CA,54:REM (SUB) READ CLOCK
390 RDY=PEEK(CB):RDY=RDYAND128:IFRDY=0GOTO390
400 DTA=PEEK(DB):POKECA,62:RETURN:REM GET DATA
```

Reading and Writing

Programs #1 and #2 provide the two basic functions of setting the time (#1) and reading the time (#2). Each has its own initialization routines. PROG #1 must set up for writing and PROG #2 must set up for reading. Normally, only one utility program for writing will be required on your disk but several programs will be reading the clock. For this reason, it may be wise to include a routine in the BEXEC* of your disk that will initialize the clock for reading when booted. Now the programs that read the clock do not require an initialization routine (assuming this has already been done). The only programs that will require initialization routines now are the set time utility, PROG #3, and any special programs for interrupt initialization. When using this scheme, any programs that initialize the clock for a write must initialize it for a read before exiting. Refer to PROG #3. The routine at line 200 sets the clock for writing on the first call (uses line 65 DATA) and sets it for reading on the second call (uses line 75 DATA). Thus, when entering the program, line 200 is called at line 90 and before exiting, line 200 is called at line 190. An alternate approach: instead of reinitializing for read at the end of the program, it could call BEXEC*, though this may not be desirable in some cases (BEXEC* could change some parameters you have set up since booting).

We now may access (read only) the clock in any program without the requirements of an initialization routine. DON'T FORGET WHEN TRANSFERRING THE PROGRAM TO ANOTHER DISK, BEXEC* OR THE ROUTINE MUST ALSO BE TRANSFERRED. Reading the clock is now reduced to

```
10 REM             PROG #3
20 REM PROGRAM TO WRITE TO TIME OF DAY CLOCK
35 REM THIS PROGRAM LEAVES THE CLOCK READY TO READ !!!!
40 REM WRITE THRU PIA
50 DA=51076:CA=DA+1:REM A DATA/A CONT
60 DB=CA+1:CB=DB+1:REM B DATA/B CONT
65 DATA 1,0,0,31,1,4,3,34,2,255,3,38
70 DATAMONTH,12,DAY OF MONTH,31,DAY OF WEEK,7,HOUR(24),23,MINUTE,59
75 DATA 1,58,0,31,1,62,3,58,2,0,3,62
80 ACODE=3:MNTH=7:REM MINUTES TO MONTHS
90 GOSUB 200:REM INITIALIZE FOR WRITE
100 ADD=17:GOSUB320:REM MASK ALL INTERRUPTS
105 :
110 FORADD=MNTHTOACODESTEP-1:READA$:READ CHK:PRINTA$;:INPUTDTA
120 IF DTA>CHK THEN RUN
130 GOSUB320:POKEDA,2:POKEDB,8:GOSUB350:NEXTADD
135 :
140 REM GO ON INPUT
150 POKE2888,0:INPUT"HIT RETURN AT THE TIME";A$:POKE2888,27
160 ADD=21:DTA=0:REM SET UP FOR GO
180 GOSUB 320:REM GO COMMAND
190 GOSUB200:END:REM SETUP FOR READ
195 :
200 REM (SUB) 1ST ENTRY SET TO WRITE/2ND SET TO READ
210 FORIN=1TO6:READIA:READIB:POKEDA+IA,IB:NEXT
220 RETURN
230 :
320 REM (SUB) WRITE TO CLOCK
325 GOSUB 370
330 REM PASS ADDRESS IN ADD,DATA IN DTA
340 POKE DA,ADD:POKE DB,DTA
350 D=PEEK(CB):D=DAND128:IFD=0 THEN 350
360 D=PEEK(DB):RETURN:REM RESET CRB-7 DATA USELESS
365 :
370 REM (SUB) CONVERT DTA FROM DECIMAL TO 2-BCD
380 IF DTA<10 THEN RETURN
390 TEMP=INT(DTA/10):LOW=DTA-(TEMP*10):TEMP=TEMP*16
400 DTA=TEMP+LOW:RETURN
```

line 380 of PROG #4. Line 390 must also be included if decimal data is required.

The read initialize routine to put in BEXEC* could be as follows:

```
2 REM INITIALIZE CA-20 FOR READ
4 DATA 1,58,Ø,31,1,62,3,58,2,Ø,3,62
6 ADD=51Ø76:FORI=1+Ø6:READIA:READIB:POKE ADD+IA,IB:NEXT.
```

PROG #4, #5, #6, and #7 assume that the clock has previously been initialized for reading by BEXEC* or PROG #3.

PROG #4 displays the time from months to seconds. Since this is a 24-hour clock, to display the time in the 12-hour format, lines 210 through 225 were included (when using PROG #3 to set the time the hours are entered in the 24-hour format).

PROG #5 gets and saves the time in decimal format in the variable array T(7)-T(Ø). This array could be used in a program to perform a specific task. In PROG #5 it is used simply to display the time in decimal.

PROG #6 is an example of a software alarm-type function (this could be done with the latch and counter hardware interrupt; direct or polled).

PROG #7 can be used to set the clock oscillator frequency if a good frequency counter is available. Connect the frequency counter to D4 (pin 19 of U1H). Run "PROG7". The frequency should be exactly 500 Hz. A constant read of the thousandths counter produces the 500 Hz square wave.

```
10 REM               PROG #4
20 REM PROGRAM TO READ TIME OF DAY CLOCK
30 FORI=1TO30:PRINT:NEXT
40 REM INITIALIZED BY BEXEC* OR PROG #3 FOR READ
50 REM READ THRU PIA
60 DA=51076:CA=DA+1:REM A DATA/A CONT
70 DB=CA+1:CB=DB+1: REM B DATA/B CONT
80 ACODE=2:MNTH=7:DIM C$(12):REM STOP AT MONTH
90 FORI=1TO12:READC$(I):NEXTI:FORI=1TO7:READD$(I):NEXTI
100 DATA JANUARY,FEBUARY,MARCH,APRIL,MAY,JUNE,JULY,AUGUST
110 DATA SEPTEMBER,OCTOBER,NOVEMBER,DECEMBER
120 DATA SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY
130 :
140 FOR ADD=MNTHTOACODESTEP-1
160 GOSUB 380:REM READ CLOCK
190 T(ADD)=TIME:NEXTADD
200 PRINTCHR$(13);D$(T(5));" ";C$(T(7));T(6);"    ";
210 IF T(4)=>12 THEN PM=1
220 IF T(4)=>13 THEN T(4)=T(4)-12
225 IF T(4)=0 THEN T(4)=12:REM 24 HOUR CLOCK ADJ
230 PRINTT(4);":"T(3)":"T(2);"  ";
240 IFPM=1THENPRINT"PM      ";:GOTO260
250 PRINT"AM      ";
260 PM=0:GOTO140:REM GO DO IT AGAIN
265 :
270 REM (SUB) READ CLOCK
380 POKEDA,ADD:POKECA,54:DTA=PEEK(DB):POKECA,62
390 MS=DTAAND240:MS=(MS/16)*10:LS=DTAAND15:TIME=MS+LS:RETURN
```

```
10 REM              PROG #5
20 GOTO230
30 :      THE SUB AT LINE 10020- COULD BE USED IN
40 :      YOUR PROGRAM TO GET THE TIME. DON'T FORGET
50 :      TO SET THE CLOCK TO THE READ MODE AS
60 :      DESCRIBED INTHE MANUAL BEFORE USING THIS
70 :
80 :      PLACE THE WEIGHT(ADD) DESIRED IN (ADD) AND
90 :      GOSUB 10020: THE TIME WILL BE RETURNED IN
100 :     DECIMAL INTHE VAR(TIME). (DTA) WILL CONTAIN
110 :     THE TIME IN PACKED BCD.
120 :     IF DECIMAL TIME IS NOT NEEDED LINE 10040
130 :     MAY BE REPLACED WITH A RETURN
140 :
150 :     THE ADDRESS(ADD) WEIGHTS ARE AS FOLLOWS:
160 :     ADD=7/MONTH            ADD=6/DAY OF MONTH
170 :     ADD=5/DAY OF WEEK      ADD=4/HOUR-24 HOUR CLCK
180 :     ADD=3/MINUTE           ADD=2/SECOND
190 :     ADD=1/HNDTHS/TNTHS     ADD=0/THOUSANDTHS
200 :
210 :     EXAMPLE (GET THE TIME AND SAVE IT)
220 :
230 :     FORADD=7TO0STEP-1      :REM SET UP WEIGHT
232 :        GOSUB 10020         :REM GO GET TIME
234 :        T(ADD)=TIME         :REM SAVE   TIME
236 :     NEXT:GOTO290           :REM UNTIL  DONE
240 :
250 :     THE VARIABLE ARRAY T(0)-T(7) NOW HAS THE TIME
260 :     AS WEIGHTED ABOVE-THE USEFULNESS OF THE HUN-
270 :     DREDTHS AND THOUSANDS IS QUESTIONABLE IN BASIC
280 :
282 :     (NOW DISPLAY THE TIME IN DECIMAL)
284 :
290 :     FORADD=7TO2STEP-1      :REM SET UP WEIGHT
292 :        PRINTT(ADD);        :REM DISPLAY IN DECIMAL
294 :     NEXT                   :REM UNTIL DONE
296 :     PRINT CHR$(13);        :REM CARRIAGE RETURN
300 :     GOTO 230               :REM GO DO IT AGAIN
390 :
400 .     >!!!THIS IS THE ONLY SECTION OF THIS !!!!!!<
420 :     >!!!   PROGRAM THAT YOU WILL NEED     !!!!!!<
10000 REM GET THE WEIGHTED TIME
10020 DA=51076:POKEDA,ADD:POKEDA+1,54:DTA=PEEK(DA+2):POKEDA+1,62
10040 MS=DTAAND240:MS=(MS/16)*10:LS=DTAAND15:TIME=MS+LS:RETURN
```

```
100 REM                   PROG #6
110 GOTO 200
120 :
130 :     GET/AN ALARM TIME: HOUR(24 HOUR CLCK) AND
140 :     MINUTE THEN TEST THE TIME FOR A MATCH
150 :     THE TIME COULD BE MATCHED FOR DATE AS WELL
160 :     AS DOWN TO THE SECOND
180 :
190 :
200 :     INPUT"HOUR(OF 24)";HOUR     :REM GET HOUR TO TEST
210 :     INPUT"MINUTE";MINUTE        :REM GET MINUTE  TEST
220 :     GOTO 280                    :REM GO CHECK FOR MATCH
230 :
240 :     TEST THE OPERATORS ALARM TIME AGAINST
250 :     THE REAL TIME UNTIL MATCH
260 :
270 :     REPEAT UNTIL (MATCH)
280 :       ADD=4:GOSUB 10020              :REM GET HOUR
290 :       IF TIME<>HOUR THEN GOTO 280    :REM HOUR MATCH?
300 :       ADD=3:GOSUB 10020              :REM GET MINUTE
310 :       IF TIME<>MINUTE THEN GOTO 280 :REM MIN  MATCH?
320 :     GOTO 380                         :MATCH FOUND
330 :
340 :     MATCH HAS BEEN FOUND TO BE TRUE
350 :     AT THIS POINT AN ALARM ROUTINE
360 :     WOULD BE ENTERED
370 :
380 :     PRINT"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
390 :     PRINT"!!!!!!!!!!!!  WAKE UP  !!!!!!!!!!!!!!!!"
400 :     PRINT"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
410 :
420 :     END:OF PROGRAM
430 :
10000 :  PROCEDURE GET THE WEIGHTED TIME
10020 DA=51076:POKEDA,ADD:POKEDA+1,54:DTA=PEEK(DA+2):POKEDA+1,6
10040 MS=DTAAND240:MS=(MS/16)*10:LS=DTAAND15:TIME=MS+LS:RETURN
```

```
100 REM                     PROG #7
110 :
120 PRINT"       !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
130 PRINT"       !!!!!!!THIS PROGRAM CAN BE USED TO:!!!!!!!!!!
140 PRINT"       !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
142 PRINT"       !!!!!!!ADJUST THE CLOCK OSCILLATOR!!!!!!!!!!!
150 PRINT"       !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
155 PRINT
160 PRINT"       PROBE (D4) OF THE CLOCK WITH A GOOD FREQUENCY
170 PRINT"       COUNTER. THE PROGRAM SETS UP A CONSTANT READ
180 PRINT"       OF THE THOUSANDTHS COUNTER WHICH SHOULD
190 PRINT"       PRODUCE A 500.000 HZ SQUARE WAVE
200 :
210 REM          (INITIALIZE) FOR READ
215 REM           INITIALIZE NOT NEEDED IF PREVIOUS INIT
220 DATA 1,58,0,31,1,1,62,3,58,2,0,3,62
230 FOR IN = 1 TO 6
240 :  READ IA
245 :  READ IB
250 :  POKE 51076+IA,IB
255 NEXT          :REM END (INITIALIZE)
257 :
260 REM          (READ CONSTANT THOUSANDTHS)
270 :
280 :   POKE 51076,0:            REM THOUSANDTHS COUNTER
290 :   POKE 51077,54:           REM READ COMMAND
300 :   END:OF:PROGRAM           DOING CONSTANT READ
```

The "GO" Command

To synchronize the clock with real time, a "GO" command exists which can be used to reset the thousandths of seconds, hundredths and tenths of seconds, and seconds counters. After setting the lower frequency counters (minutes through months), the appropriate address (21 decimals) and a write pulse can be sent to reset all counters mentioned above. This allows the clock to be started at an exact known time. It can also be used as a stopwatch function. The "GO" command is the start and a counter read is the stop point. The clock does not stop during or following a read, so each read would be a split time. PROG #1 and PROG #3 uses this function.

The Real Time Clock

The circuit includes addressable real time counters and addressable latches each for thousandths of seconds through months. The counters and latches are divided into bytes of four bits each. When addressed, two bytes will appear on the data I/O (input/output) bus. The data, in binary coded decimal (BCD), can be transferred to and from the counters via the data I/O bus so that each set of two bytes (one word) can be accessed independently. Available are thousandths of seconds, hundredths of seconds, tenths of seconds, seconds, minutes, hours, day of the week, day of the month, and month counters with corresponding latches for alarm-type functions. Interrupts available (hard wired or polled) are latch and counter comparison, every tenth of a second, every second, every minute, every hour, every day, every week, and every month. Also a standby interrupt output is available. This output can be used to power cycle specially equipped systems using latch and counter comparisons. To support this feature, other functions are available on the CA-20. These are discussed in another section of this manual.

Four Year Calendar

The clock always "rolls" the month at midnight, February 28. Therefore, each leap year, the clock must be reset.

Standby Interrupt

The standby interrupt output is the only input or output enabled during the power down mode (system power off). This signal would only be operational in specially equipped systems that have a "smart power sequencer". That is a current sink driver which is available on the CA-20 that could "FLAG" a power sequencer package to turn on the computer. This output could also be used to power down the computer under software control. Once set, the output is "TRUE" until reset by writing to the interrupt control register disabling it. Thus, an unattended system could power itself up at a predetermined time, perform a task, and shut itself off. This feature would also require a modification to the "firmware" as described in a later section.

Refer to Figure 3 (Interrupt Register Format) for interrupt mask control register information.

Interrupts

The interrupt output is controlled by the interrupt status register (eight bits) and the interrupt control register (eight bits). See Figure 3 . The status register contains the present state of the comparator (compares the counters and latches) and the outputs (one bit each) of the tenths of seconds, second, minutes, hours, week, day of the month, and month counters. The interrupt status register can only be read. The interrupt control register is a mask register that regulates which of the eight bits in the status register goes out as an interrupt. The control register cannot be read from. A one (1) is written into the control register to select the appropriate interrupt output. If more than a single one exists in the control register each selected bit will come out as an interrupt. This will appear as an interrupt occurring at the highest frequency selected. The interrupt is seen as a rising edge to CA1 of the PIA, which must be initialized to "pass" the interrupt to the system bus if desired. The interrupt can be hard wired to the bus or could be used as a polled interrupt by polling CRA-7 of the PIA. CRA-7 is acknowledged by reading DATA(A) (51Ø76) which resets CRA-7. The interrupt from the clock is acknowledged by addressing and reading the status register. Once acknowledged, the interrupt output and status register are reset. The only way to disable the interrupt output from the clock to the PIA is to write all zeros into the control register. Of course, the PIA could be re-initialized to disable IRQ outputs.

Batteries

The batteries for standby operation of the real time clock (included with the CA-20) are charged before shipment. Due to normal self-discharge characteristics of nickel-cadmium batteries, they may require charging after the initial installation before an extended standby operation (power off over an extended period of time). The batteries are recharged automatically by the CA-20 whenever the system power is on. A full charge cycle would consist of approximately 12-hours with the system power on (6-hours for two days). Fully charged batteries should operate the clock in the standby mode greater than one month. For applications that could conceivably require longer standby operation, provisions are made for connecting to user supplied (remote) standby batteries of greater capacity than can be placed on the circuit board. The batteries supplied with the CA-20 should have a useful life span... from three to five years.

If replacement becomes necessary, only premium quality (AA) sealed nickel-cadmium batteries should be used. Lead-acid or alkaline cells, or any battery designated (DO NOT RECHARGE) should never be used unless the charging action of the CA-20 is defeated.

Clock Adjustment

The RTC (real time clock) on the CA-20 board may from time to time require a slight adjustment of the crystal trimmer capacitor (C2) if it appears to be gaining or losing time. Refer to Figure 2. The trimmer capacitor can be adjusted by using a small jewelers screwdriver or your fingernail. Two separate methods could be used to "TUNE" the oscillator.

Frequency Measurement

Connect the probe of a frequency counter to D4 of the clock chip (a vector pin is provided for this). Run PROG #7. C2 should now be adjusted until a 500.000 Hz square wave is verified. The instrument (screwdriver, etc.) used to adjust the trimmer capacitor must be removed each time a reading is taken.

Time Measurement

Adjust the trimmer in the direction required (Figure 2). A small degree of rotation should be performed. Check the time after a period of at least 24-hours to evaluate the results.

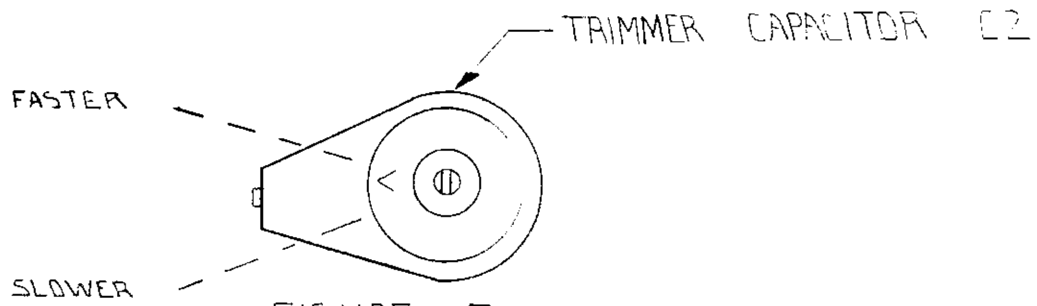


FIGURE 2

TABLE II

ADDRESS CODES AND FUNCTIONS

A4	A3	A2	A1	AØ	FUNCTION
Ø	Ø	Ø	Ø	Ø	Counter - Thousandths of Seconds
Ø	Ø	Ø	Ø	1	Counter - Hundredths and Tenths of Seconds
Ø	Ø	Ø	1	Ø	Counter - Seconds
Ø	Ø	Ø	1	1	Counter - Minutes
Ø	Ø	1	Ø	Ø	Counter - Hours
Ø	Ø	1	Ø	1	Counter - Day of the Week
Ø	Ø	1	1	Ø	Counter - Day of the Month
Ø	Ø	1	1	1	Counter - Months
Ø	1	Ø	Ø	Ø	Latches - Thousandths of Seconds
Ø	1	Ø	Ø	1	Latches - Hundredths and Tenths of Seconds
Ø	1	Ø	1	Ø	Latches - Seconds
Ø	1	Ø	1	1	Latches - Minutes
Ø	1	1	Ø	Ø	Latches - Hours
Ø	1	1	Ø	1	Latches - Day of the Week
Ø	1	1	1	Ø	Latches - Day of the Month
Ø	1	1	1	1	Latches - Months
1	Ø	Ø	Ø	Ø	Interrupt Status Register
1	Ø	Ø	Ø	1	Interrupt Control Register
1	Ø	Ø	1	Ø	Counter Reset
1	Ø	Ø	1	1	Latch Reset
1	Ø	1	Ø	Ø	Status Bit
1	Ø	1	Ø	1	"GO" Command
1	Ø	1	1	Ø	Standby Interrupt

All others unused.

TABLE III

COUNTER AND LATCH RESET FORMAT

DØ	D1	D2	D3	D4	D5	D6	D7	COUNTER OR LATCH RESET
1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Thousandths of Seconds
Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Hundredths and Tenths of Seconds
Ø	Ø	1	Ø	Ø	Ø	Ø	Ø	Seconds
Ø	Ø	Ø	1	Ø	Ø	Ø	Ø	Minutes
Ø	Ø	Ø	Ø	1	Ø	Ø	Ø	Hours
Ø	Ø	Ø	Ø	Ø	1	Ø	Ø	Days of the Week
Ø	Ø	Ø	Ø	Ø	Ø	1	Ø	Days of the Month
Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	Months

FOR COUNTER RESET A4— AØ MUST BE 1ØØ1Ø

FOR LATCH RESET A4— AØ MUST BE 1ØØ11

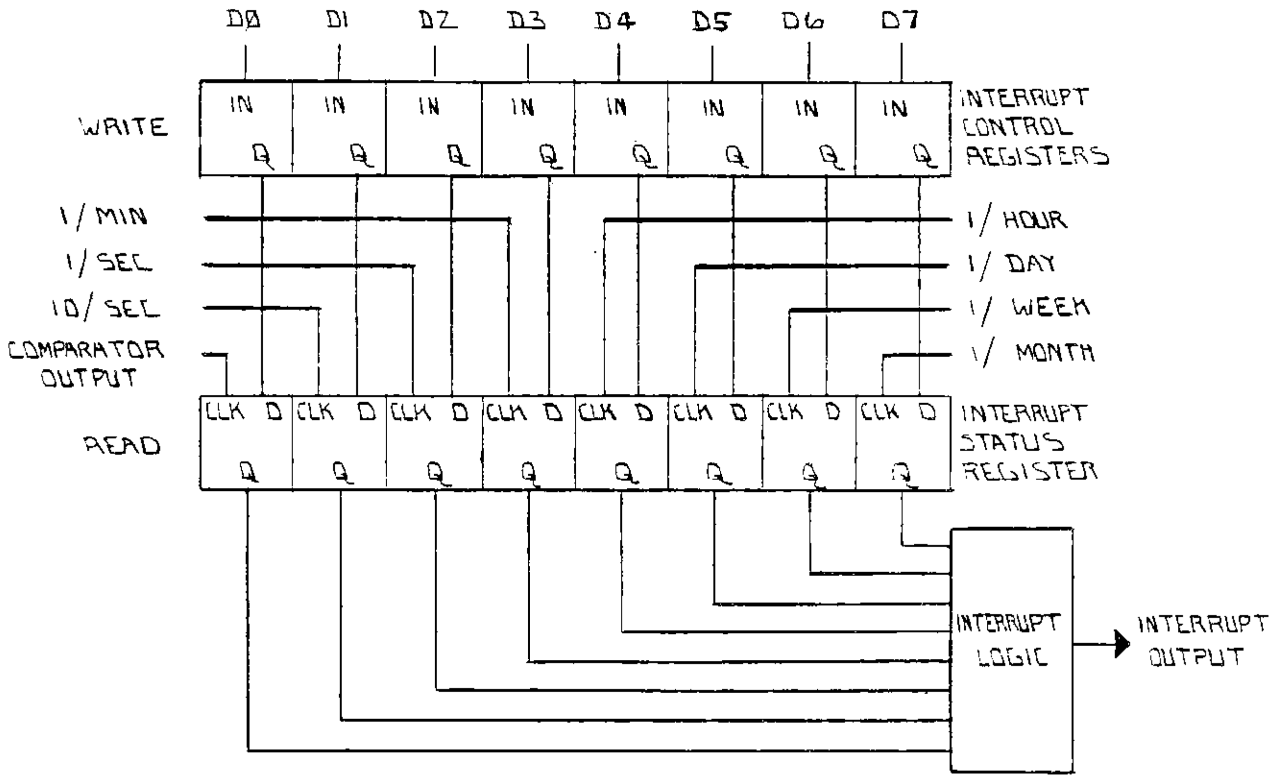


FIGURE 3

TABLE IV

MAXIMUM USED BCD CODE

	UNITS				MAX USED	TENS				MAX USED
COUNTER ADDRESSED	DØ	D1	D2	D3	BCD CODE	D4	D5	D6	D7	BCD CODE
Ten Thousandths of a Second	Ø	Ø	Ø	Ø	Ø	I/O	I/O	I/O	I/O	9
Tenths and Hundredths of Seconds	I/O	I/O	I/O	I/O	9	I/O	I/O	I/O	I/O	9
Seconds	I/O	I/O	I/O	I/O	9	I/O	I/O	I/O	Ø	5
Minutes	I/O	I/O	I/O	I/O	9	I/O	I/O	I/O	Ø	5
Hours	I/O	I/O	I/O	I/O	9	I/O	I/O	Ø	Ø	2
Day of the Week	I/O	I/O	I/O	Ø	7	Ø	Ø	Ø	Ø	Ø
Day of the Month	I/O	I/O	I/O	I/O	9	I/O	I/O	Ø	Ø	3
Month	I/O	I/O	I/O	I/O	9	I/O	Ø	Ø	Ø	1

SPECIAL HARDWARE

CA-20

Refer to Sheet 1 of the schematics. The CA-20 is equipped with special circuitry to perform the following:

WATCHDOG TIMER
NMI ON PWR UP
SYSTEM RESET ON PWR UP
STANDBY INTERRUPT/AUTO CYCLE
POWER FAIL RESET

Switch

A DIP (dual in-line pole) switch is used to configure the CA-20 to one of many possible operating modes. The pins of the switch are designated 1 through 16. In this discussion, the eight switches will be referred to as SW1-SW8 (SW1 includes pins 1 and 16; SW8 includes pins 8 and 9).

NMI

SW4 enables the non-maskable interrupt to the processor. This should only be used in systems designated by the factory. NMI can be used (with special firmware unless designated as such - NOT PRESENT) to auto boot the system on power up. This could be used in conjunction with the standby auto cycle feature for unattended power cycling of systems or turnkey bootstrap, or auto restart power fail situations. With SW1 in the enable position, NMI occurs after a reset and a delay period determined by U3I and U1F. Normally this would be the only NMI in the system so software polling is not required. In the case of multiple NMI sources, NMI may be read at 51Ø76, bit 7. The PIA will not require initialization after a reset since this is a data input. NMI is reset (disabled) by writing a one (1) to D2 of $C7AØ (511Ø4). If either SW2 or SW3 is true, NMI is enabled by the system reset. This register also contains other function enable/disable bits which must be observed (SW2/SW3 must never be true at the same time). NMIEN (non-maskable interrupt enable) at J8, pin 5, is a remote input to enable/disable NMI. SW1 can override this function.

System Reset

System Reset, J8, pin 3, is an (or) function with the 510 board. Reset can be generated by the front panel switch, the CA-20 output or the processor select signal on the 510. The application will determine whether the last signal should be included. The system reset is generated by the CA-20 under several circumstances and options. Power on reset is always generated by the C13, R9 circuit. Power down/fail reset is always generated by the standby/ power fail circuit Q2/Q3 and discrete circuitry. The other two resets are optional and can be controlled by manual switches or software. These are the "watchdog timer" and the "sync detector".

Watchdog Timer

The watchdog timer is manually enabled by SW6. The circuitry consists of a resetable timer (U1F, U3I), watchdog reset control (U3E, U3H), and an enable control (U3F, SW6). The watchdog enable (U3F) is reset by any system reset and must be enabled by a write to $C7AØ (511Ø4). A (1) on D1 enables and a (Ø) disables the watchdog. After enabling, the watchdog must be signaled that the program is "running" by a write to $C7CØ (51136) at a rate at least slightly faster than the timeout period of the timer (U1F, pin 1). If the watchdog timer times out, a system reset will be generated. This would cause all to be set and if so equipped/

desired, a new "boot" would take place.

Sync Detector

The 6502 processor generates a "SYNC" signal each time it does an instruction fetch. The sync detector monitors these sync pulses to determine if the processor is still running. This is enabled by writing to $C7AØ (511Ø4). A (1) on DØ enables it and a (Ø) disables missing sync detection. SW5 enables missing sync detection to generate a system reset pulse. The enable is reset on system reset. If either of the other two processors on the 510 are selected, the missing sync detector must be disabled.

SPECIAL FUNCTIONS

		D7	D6	D5	D4	D3	D2	D1	DØ
$C7AØ	511Ø4	X	X	X	X	X	Ø=ENA NMI	ENA WATCHDOG	ENA SYNC
$C7CØ	51136	X	X	X	X	X	X	X	X

$C7CØ=WATCHDOG "TIC" - DATA IRRELEVANT

BOTH ARE WRITE ONLY

FUNCTION	SW8	SW7	SW6	SW5	SW4	SW3	SW2	SW1
$\overline{\text{NMI}}$	X	X	X	X	C	Ø	C	Ø
WATCHDOG	X	X	C	X	X	Ø	C	X
SYNC	X	X	X	C	X	Ø	C	X
NMIEN	X	X	X	X	C	Ø	C	Ø/J8, pin 5
R.T.C. IRQ	X	C	X	X	X	Ø	C	X

Installation

CAUTION: Remove power from the computer before proceeding with the following steps. Remove the screws that fasten the cover to the computer, then carefully remove the cover and set aside. On some C2-C3 systems the Molex power connector to the muffin fan(s) must be disconnected before complete removal of the cover.

The ribbon cables from/to the "HEAD END CARDS" should be routed through the floppy/hard disk cable access hole (C8P, C2, C3) at this time. If installing in a C4P the cables should be routed in the same manner as the floppy cable (through the point where the cover screws to the frame). Do not tighten these screws excessively when reinstalling the cover.

Referring to Figure 1, connect the ribbon cables to the CA-20. It is recommended that the first "HEAD END CARD" ribbon cable be connected to Port Ø, then as more are added in the future they be connected to Ports 2, 4, 6, 1, 3, 5, 7 in that order. Refer to the "HEAD END CARD" manual for power supply connections/requirements. Refer to the Power Distribution section of this manual for power connections to your "HEAD END CARDS".

Find or create (by moving another card) a free slot close enough to the ribbon cable access hole to give maximum or necessary ribbon cable lead length outside of the case and install the CA-20 in this slot. Be very careful to align the 48 pin bus connector correctly as damage could result from faulty installation.

Return the cover to the case in the reverse order of removal, making sure that any ground straps or fans, if any, are reconnected.

Installation of the CA-20 in the C4P MF, C4P DF or C8P DF which are 505 Rev. B (CPU) based requires that the accessory bus

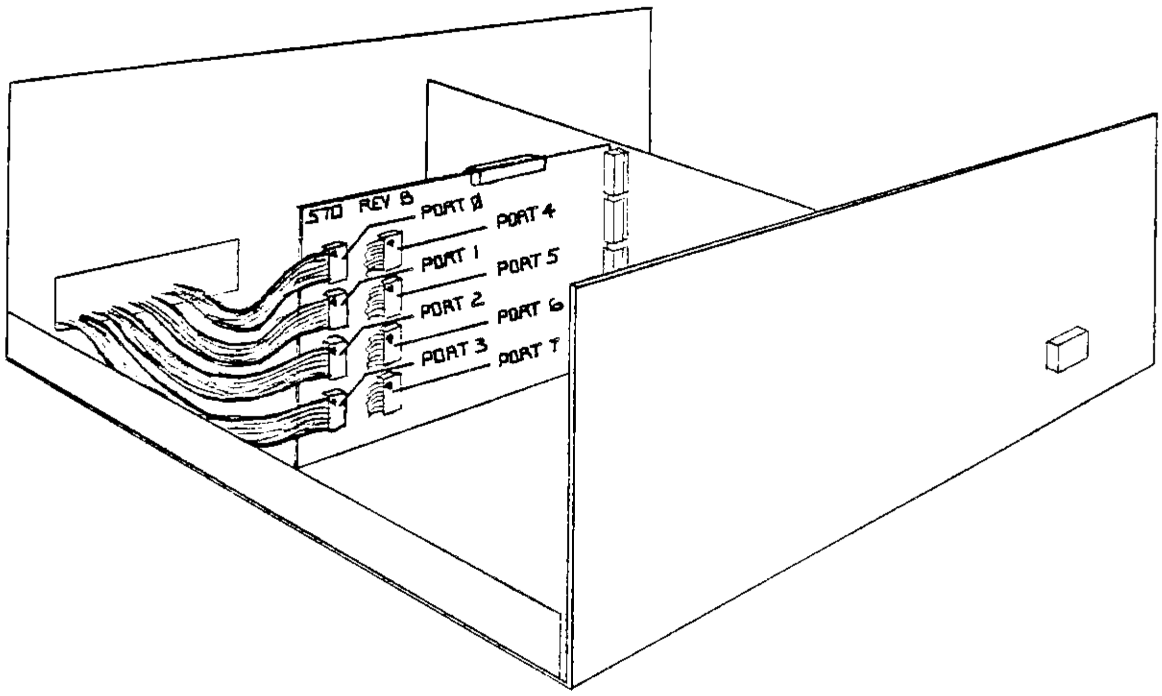


FIGURE 1

interface on the 505 Rev. B be disabled. This is very important since improper operation of the CA-20 will be experienced if not done. Remove the 505 Rev. B from the card cage. Pay close attention (a quick sketch would be helpful) to any connectors that may have to be removed to get the 505 Rev. B in a suitable position for the following actions. Referring to Figure 2, remove the integrated circuit packages U4H and U3G. These two packages should be labeled (8T26 or 75136). This can be accomplished with a small screwdriver or similar instrument. Alternately pry up on each end (short end with no pins) of the package until it has been "rocked" out of the socket. Be sure that the screwdriver is placed between the package and the socket and not between the socket and the printed circuit board. You should find this to be a very simple operation. Alternately the accessory bus on the 505 Rev. B could be readdressed moving the enable line from U5J Pin 15 to U5J Pin 14 ($C6XX). This is not recommended since this is an OSI reserved address block and could possibly cause problems if any upgrades are ever added to your system. This type of modification should be left to an experienced technician with the proper tools and knowledge. If a "HEAD END CARD" was previously connected to the A-15 card on the back of your computer, it should now be connected to Port Ø on the CA-20. Software for that card will still be fully compatible. The other interface operations of the A-15 will be unaffected and can still be used as explained in your manual. When this is completed re-install the 505 Rev B in the computer taking care to replace any connectors previously removed in the exact locations noted before.

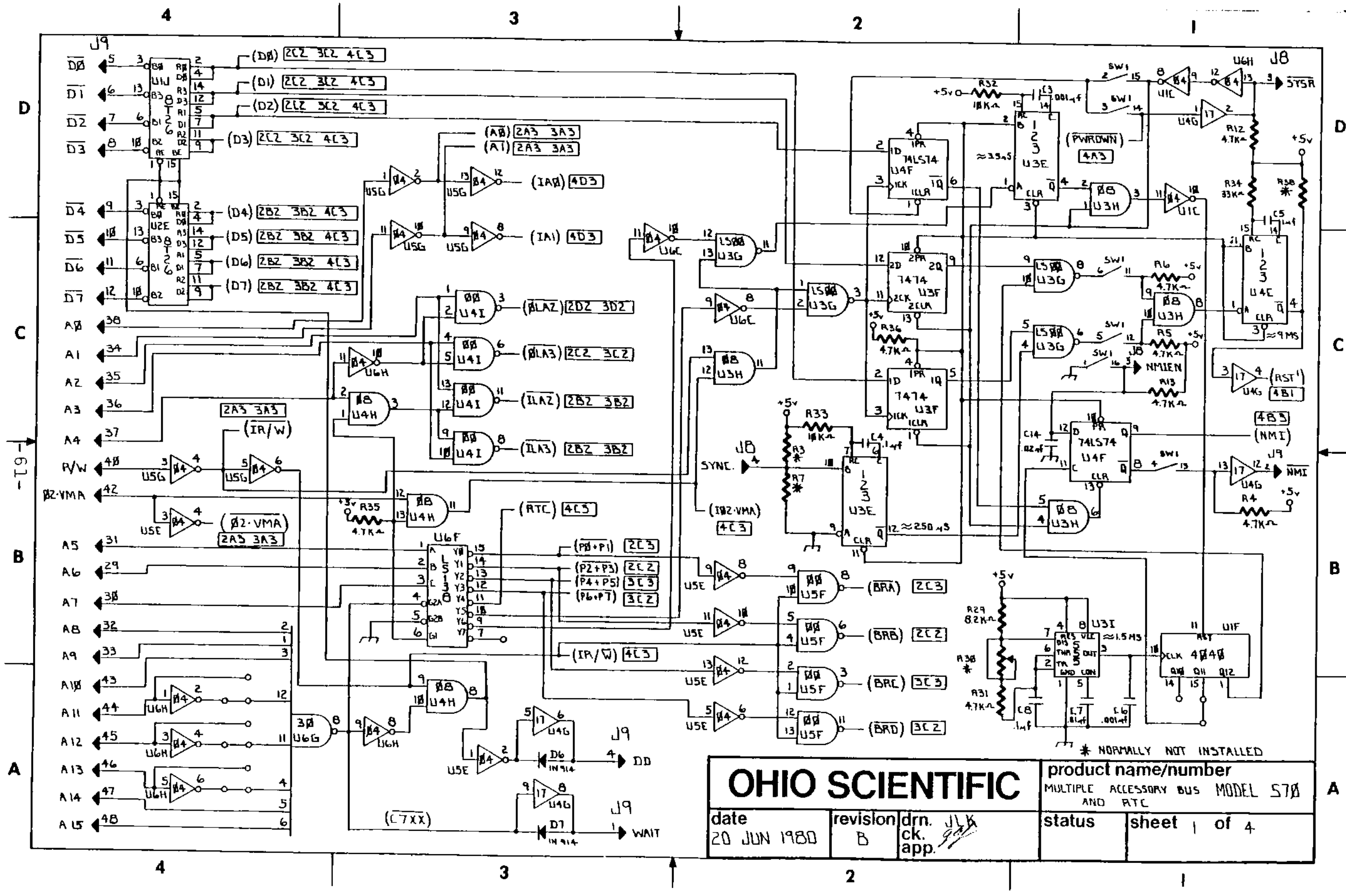
Return the cover to the case in the reverse order of removal making sure that any ground straps or fans, if any, are reconnected.

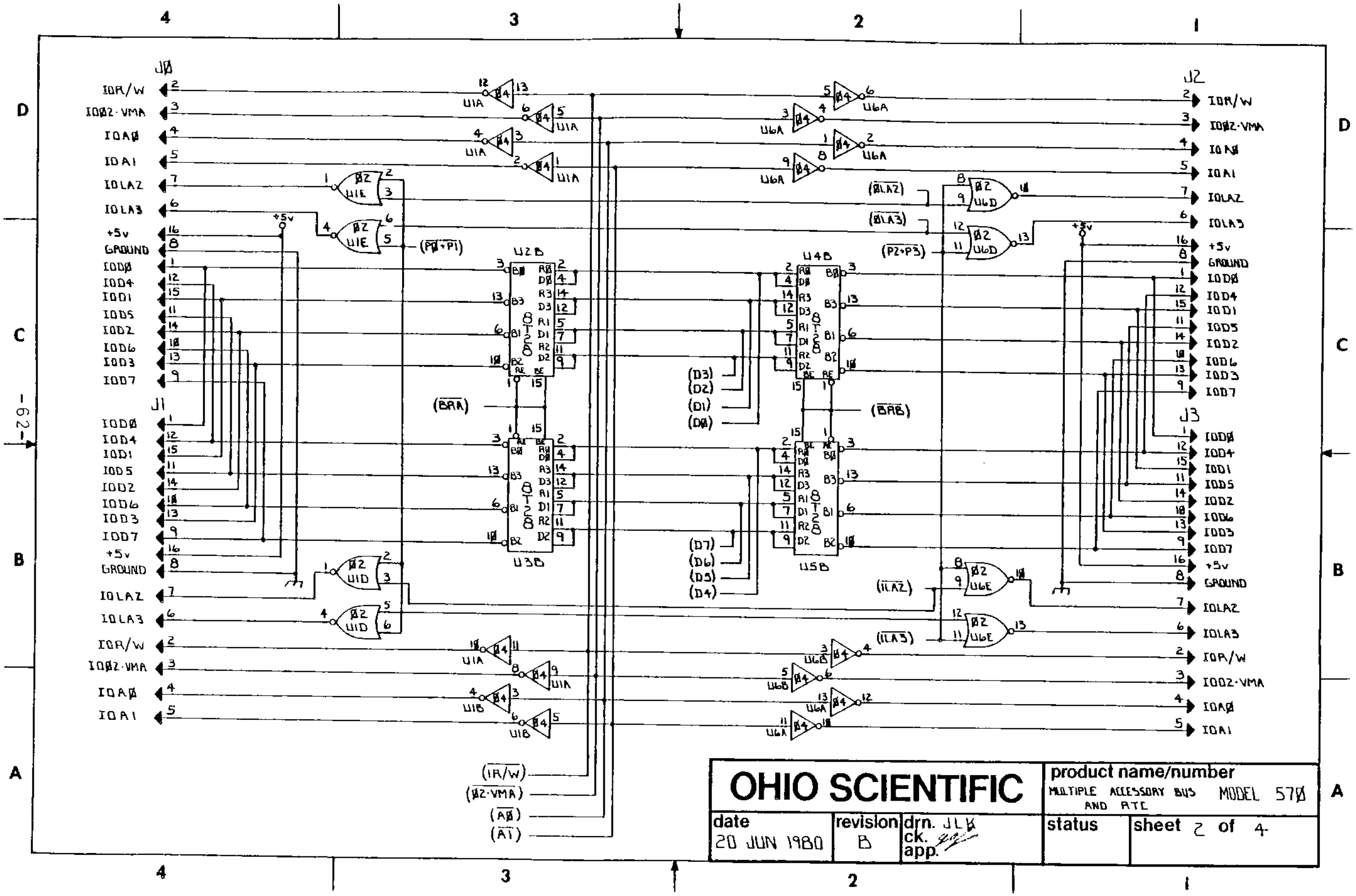
Power Distribution

Each 16 pin ribbon cable "HEAD END CARD" interface port on the CA-20 has a digital ground reference and a +5V supply pin (Pins 8 and 16 respectively). These may be used to power some of the "HEAD END CARDS" directly. Refer to the "HEAD END CARD" manual for each board interfaced to determine whether power should be supplied directly through the 16 pin ribbon cable or by alternate means. Examples would be the CA-21 and CA-25 which are supplied through ribbon cables. To interface these simply plug in the ribbon cable. On other boards such as the CA-22 (analog interface) and CA-24 (solderless prototyping board) the power must be supplied via alternate means. On these "HEAD END CARDS" Pin 16 has been left open and a "HEAD END CARD" standard power connection is provided. Power to the card should be supplied via this connection from either an auxiliary supply or routed back to the system supply. In the case where system power will be routed to the "HEAD END CARD" via these connectors from the system supply, solder pads have been provided near the edge of the CA-20 board for VCC (+5V) and ground. If system power is tapped, extreme care must be observed to prevent overloading the system supply. Also, if several directly powered devices such as the CA-21 are installed, system power supply capabilities must be observed. The best approach when installing multiple accessory boards or any one accessory board with high current requirements is an auxiliary supply dedicated to the accessory cards. The directly powered "HEAD END CARDS" such as the CA-21 and CA-25 have been supplied with the capability of auxiliary power supply connections. Under these circumstances

Pin 16 of the ribbon cable is cut and +5V and ground are routed through solder pads or connectors. Refer to each individual "HEAD END CARD" manual.

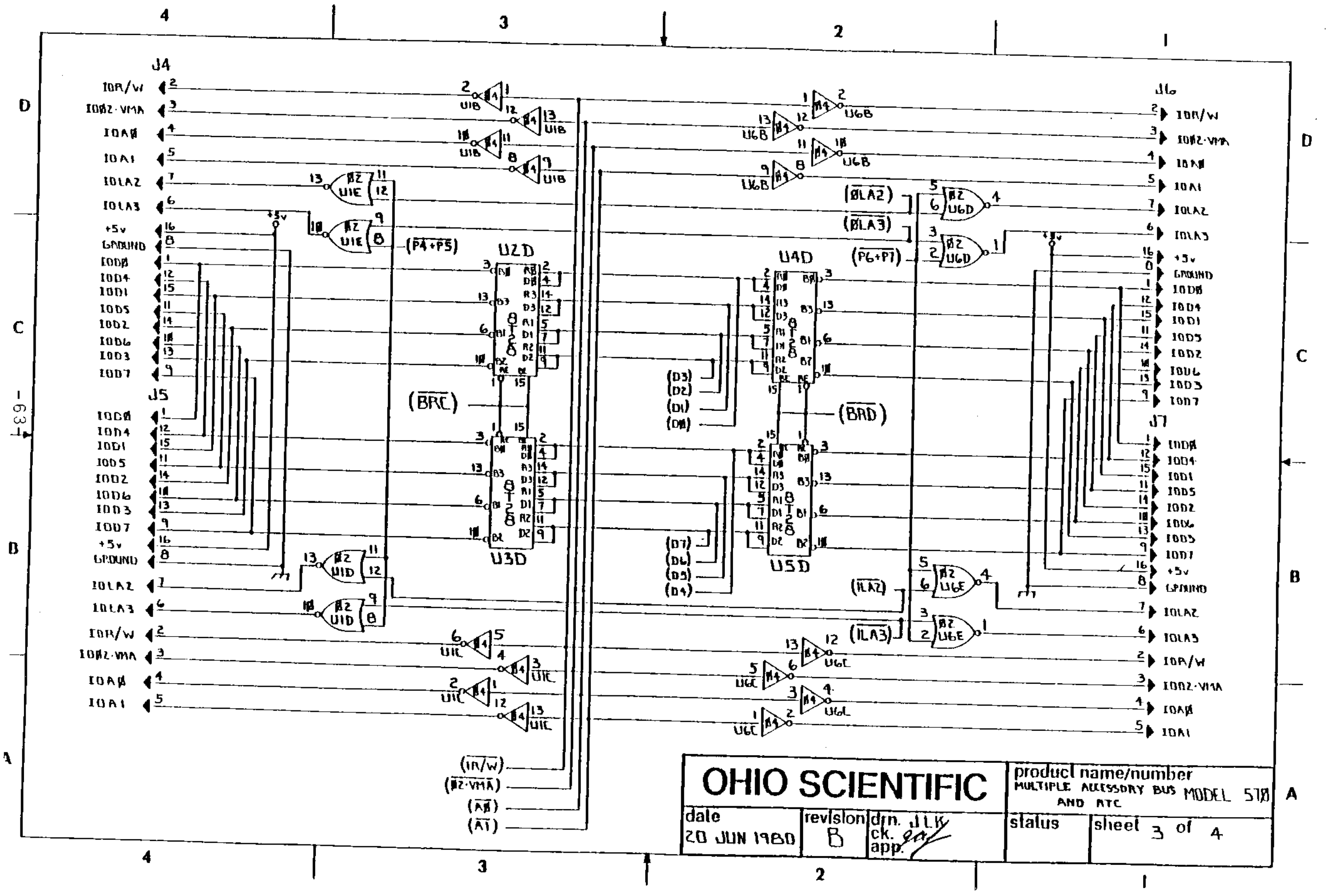
```
10 GOTO 80 :                  PROG #B
20 :       READ TIME OF DAY CLOCK (CA-20)
30 :       SAME AS (PROG #4)
40 :       THIS PROGRAM ASSUMES THAT THE CLOCK
50 :       HAS BEEN INITIALIZED FOR READING
60 :       BY MODIFIED (BEXEC*) OR (PROG #3)
70 :
80 FOR SCROLL = 1 TO 30           :REM CLEAR SCREEN
90 :  PRINT
100 NEXT SCROLL                   :REM SCREEN IS CLEAR
110 :
120 :        REM (DEFINE) ADDRESSES OF CLOCK
130 :
140 DA=51076:CA=DA+1:REM A DATA/A CONT
150 DB=CA+1:CB=DB+1: REM B DATA/B CONT
160 MINUTE=2:MNTH=7:DIM MNTH$(12) :REM START-MINUTE/STOP-MONTH
170 DATA JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC
180 DATA SUN,MON,TUE,WED,THU,FRI,SAT
190 :
200 :       REM (FILL DISPLAY) ARRAY
210 FOR I = 1 TO 12               :REM GET 12 MONTHS
220 :  READ MNTH$(I)              :REM STUFF IN ARRAY
230 NEXTI                         :REM LOOP UNTIL DONE
240 FOR I = 1 TO 7                :REM GET 7 DAY/WEEK
250 :  READ DAY$(I)               :REM STUFF IN ARRAY
260 NEXTI                         :REM LOOP UNTIL DONE
270 :
280 :       REM (FILL TIME) ARRAY W/CURRENT TIME
290 FOR ADD = MNTH TO MINUTE STEP-1
300 :  GOSUB 500                  :REM READ CLOCK
310 :  TIME(ADD)=TIME             :REM STUFF ARRAY W/TIME
320 NEXT ADD                      :REM LOOP UNTIL DONE
330 :
340 :       REM (DISPLAY TIME)
350 PRINTCHR$(13);                 :REM CARRIAGE RET. NO LF
360 PRINT DAY$(TIME(5));" ";       :REM PRINT DAY OF WEEK
370 PRINT MNTH$(TIME(7));          :REM PRINT MONTH NO CRLF
380 PRINT TIME(6);"   ";           :REM PRINT DAY OF MONTH
390 IF TIME(4)=>12 THEN PM=1       :REM ADJ FOR 24 HOUR CLOCK
400 IF TIME(4)=>13 THEN T(4)=T(4)-12
410 IF TIME(4)=0 THEN TIME(4)=12 :REM 24 HOUR CLOCK ADJ
420 PRINTTIME(4);                 :REM PRINT HOUR
430 PRINT TIME(3)":";             :REM PRINT MINUTE
440 PRINT TIME(2);"  ";           :REM PRINT SECOND
450 IFPM=1THENPRINT"PM      ";:GOTO470
460 PRINT"AM      ";
470 PM=0:GOTO280                  :REM GO DO IT AGAIN
480 :
490 :           (SUB) "READ CLOCK"
500 POKEDA,ADD                    :REM SET UP CLOCK ADDRESS
510 POKECA,54                     :REM SEND READ COMMAND TO CLOCK
520 DTA=PEEK(DB)                  :REM GET THE WEIGHTED TIME
530 POKECA,62                     :REM REMOVE READ COMMAND
540 :
550 REM    (DTA)=BCD TIME   :      NOW CONV TO DECIMAL
560 MS=DTAAND240                  :REM GET MS-BYTE
570 MS=(MS/16)*10                 :REM CONVERT TO DECIMAL
580 LS=DTAAND15                   :REM GET LS-BYTE
590 TIME=MS+LS                    :REM MERGE TIME=DECIMAL TIME
600 RETURN  : END OF-READ CLOCK    :TIME=DEC/DTA=BCD
```


OHIO SCIENTIFIC
product name/number
MULTIPLE ACCESSORY BUS AND RTC
MODEL 57Ø
date 20 JUN 1980
revision B
drn. JLK
ck.
app.
status
sheet 1 of 4
✱ NORMALLY NOT INSTALLED
J9
J8
SYSR
NMI
NMIEN
SYNC.
DD
WAIT
(PWRDWN)
(RST')
(NMI)
(RTC)
(C7XX)
(IR/W)
(Ø2·VMA)
74LS74
7474
4Ø4Ø
555
U6F



OHIO SCIENTIFIC

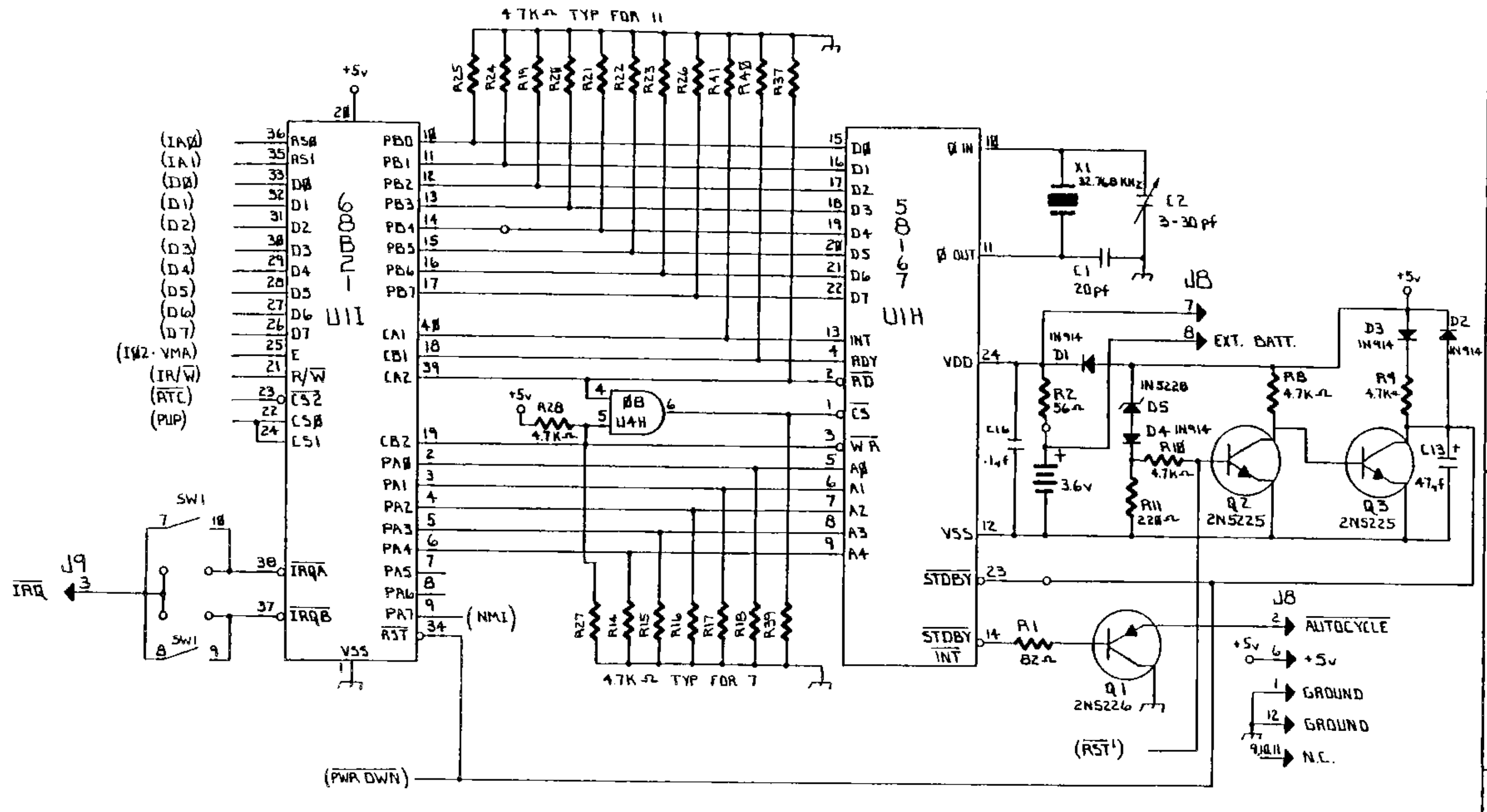
date 20 JUN 1980	revision B	drn. JLK ck. [illegible] app.	product name/number MULTIPLE ACCESSORY BUS AND RTC MODEL 57Ø	
			status	sheet 2 of 4



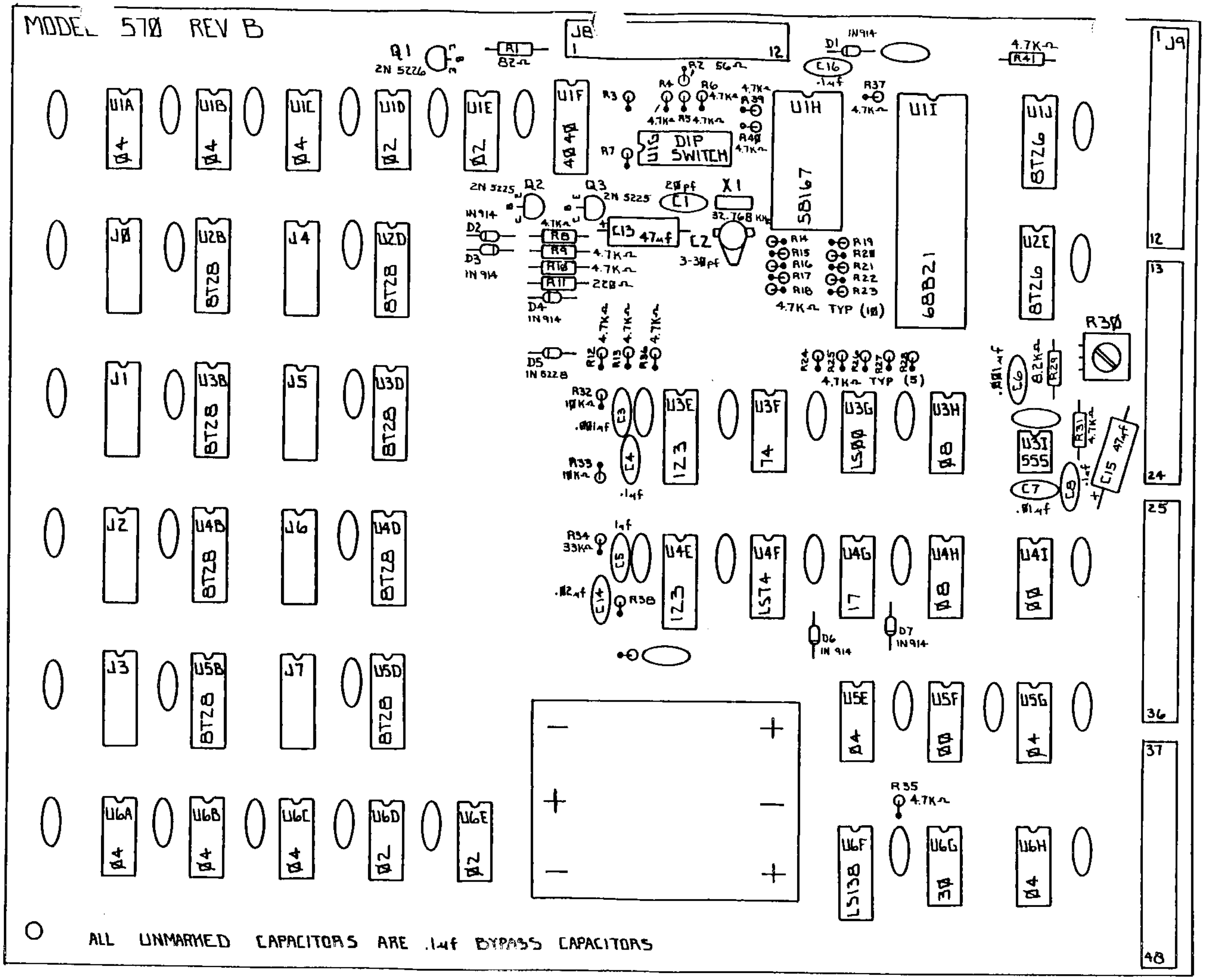
OHIO SCIENTIFIC

product name/number
MULTIPLE ACCESSORY BUS AND RTC MODEL 51Ø

date	revision	drn. JLK ck. app.	status	sheet
20 JUN 1980	B			3 of 4


4.7KΩ TYP FOR 11
R25 R24 R19 R20 R21 R22 R23 R26 R41 R40 R37
+5v
6821
U11
RS0 RS1 D0 D1 D2 D3 D4 D5 D6 D7 E R/W CS2 CS0 CS1
PB0 PB1 PB2 PB3 PB4 PB5 PB6 PB7 CA1 CB1 CA2 CB2 PA0 PA1 PA2 PA3 PA4 PA5 PA6 PA7 RST
IRQA IRQB VSS
(IA0) (IA1) (D0) (D1) (D2) (D3) (D4) (D5) (D6) (D7) (IØ2·VMA) (IR/W) (RTC) (PUP)
SW1
J9
IRQ
(NMI)
(PWR DWN)
R28 4.7KΩ
0B U4H
R27 R14 R15 R16 R17 R18 R39
4.7KΩ TYP FOR 7
5867
U1H
D0 D1 D2 D3 D4 D5 D6 D7 INT RDY RD CS WR A0 A1 A2 A3 A4
Ø IN
Ø OUT
VDD
VSS
STDBY
STDBY INT
X1 32.768 KHz
C2 3-30pf
C1 20pf
J8
EXT. BATT.
1N914 D1
R2 56Ω
C16 .1μf
3.6v
1N5228 D5
D4 1N914
R10 4.7KΩ
R11 220Ω
R8 4.7KΩ
Q2 2N5225
D3 1N914
D2 1N914
R9 4.7KΩ
Q3 2N5225
C13 47μf
R1 82Ω
Q1 2N5226
(RST')
AUTOCYCLE
+5v
GROUND
GROUND
N.C.

OHIO SCIENTIFIC			product name/number MULTIPLE ACCESSORY BUS AND RTC MODEL 570	
date 20 JUN 1980	revision B	drn. JLK ck. [illegible] app.	status	sheet 4 of 4



ALL UNMARKED CAPACITORS ARE .1uf BYPASS CAPACITORS