RCA MICROPROCESSOR 1800

Operator Manual for the RCA COSMAC Development System II CDP18S005

MPM-216 Suggested Price $10.00

Operator Manual for the RCA COSMAC Development System II CDP18S005

RCA Solid State

Buenos Aires • Hamburg • Liege • Madrid • Mexico City • Milan
Montreal • Paris • Sao Paulo • Somerville NJ • Stockholm
Sunbury-on-Thames • Taipei • Tehran • Tokyo

Printed in USA/10-77

Foreword

The COSMAC Development System CDP18S005 is a prototyping aid for the design of hardware and software systems based on the RCA CDP1802 microprocessor. The COSMAC Development System is specially structured to provide a testbed in which hardware/software prototypes of systems containing a microprocessor may be designed, built, and tested. In small-volume applications it can be used as the major building block for dedicated microcomputers.

This Manual is designed as a guide for the COSMAC Development System user. It includes a detailed description of each of the available hardware modules as well as a complete explanation of the functions available from the software supplied with the system.

The COSMAC Development System (CDS) consists of a card nest with self-contained power supply, an easy-to-use control panel, and a basic set of plug-in modules. It is packaged to promote easy interfacing with external devices. These interfaces may be custom-designed by the user or, in the case of common peripheral devices, are available from RCA as standard optional modules, and include a floppy disk interface.

The COSMAC Resident Software Package (which runs on the CDS in a stand-alone manner) provides a means for rapid coding and debugging of COSMAC programs. Many of its features are compatible with those of the COSMAC Software Development Package (CSDP) timesharing program-development aids. Additional software and firmware packages are available from RCA including packages for floppy disk hardware and multiple precision arithmetic.

Table of Contents

Operating and Programming the CDS

System Overview

The CDP18S005 COSMAC Development System (CDS) consists of a power supply, control panel, and a set of connectors for printed circuit boards. Many of the 25 available positions are occupied by specific module types. A printed circuit backplane distributes common signal lines to all connector positions. There are a small number of additional wire-wrapped connections. The unassigned connectors are available for user expansion of memory and I/O function.

Supplied modules include the CDP1802 CPU module, an address latch and memory bank select module, a 4-kilobyte RAM module, a ROM/RAM module containing the Utility program, an I/O decode module, terminal interface module, and a control module. The position assignments of these modules are given in Table III in the next Section. All logic functions are implemented in CMOS operating at + 5 V.

The control panel provides a simple user interface. Depressing the RESET switch initializes the system. Depressing the RUNU switch starts the utility program, identified as UT20. Depressing RUNP, on the other hand, will start program execution from memory location 0, the normal starting location of a user program. A STEP/CONTINUOUS switch allows stepping one machine cycle with each depression of RUNU or RUNP. A 4-digit display on the front panel shows the current address and a 2-digit display shows the value of the data bus or, by switch selection, the last I/O data byte. Five additional LED indicators monitor the State Code, WAIT, CLEAR, and Q lines of the CPU while a sixth LED indicates when the machine is running. This RUN indicator will be ON whenever the CPU is running and not in the IDLE mode. The LOAD switch is used to place the CPU in the 'load mode' in which the DMA channel can be used to load RAM. See **The User Manual for the CDP1802 COSMAC Microprocessor**, MPM-201, for a detailed discussion of this mode. The LOAD switch is supplied as a convenience for the user designing his own system interfaces and is not used in normal CDS operations. The supplied programs are loaded using the Monitor program as discussed in the next section.

The CDS is designed to work with any one of the following terminals:

1) An ASR 33 Teletype♦ terminal (or its electrical equivalent) which should include a remote reader control circuit to permit the CDS to control the paper tape reader.

2) A TI "Silent 700" terminal▲, Model 733 ASR with tape cassettes and "Remote Device Control" option. This terminal uses dual program-controlled magnetic tape cartridges as storage medium and prints at 30 characters per seconds.

3) Any terminal conforming to the EIA RS232C standard interface and having a baud rate of 110, 300, or 1200.

The CDS is designed to automatically adjust to a variety of data terminal speeds and will accommodate either full- or half-duplex operation.

Included with the CDS are an assembler and editor program for software development. Loading instructions for these programs are given in the next subsection. Details on operation of the Assembler and Editor are given later in this manual. If a Floppy Disk system is used with the CDS, refer to the Floppy Disk Manual for operation of the Assembler, Editor, and other programs.

♦ Registered Trademark, Teletype Corporation.
▲ Registered Trademark, Texas Instruments Corporation.

Initial Operation

CDS Hookups

Two I/O data terminal cables are supplied with the Development System - one for terminals using a 20-mA loop interface, the other for an EIA RS232C data terminal. Each cable with its terminal board connection is labeled. To connect the data terminal, select the proper cable and plug its receptacle labeled "P1" into the appropriate connector mounted directly on the accessible end of the terminal board, as shown in Fig. 1. For a list of terminal interface cables and their functions, refer to Appendix J.

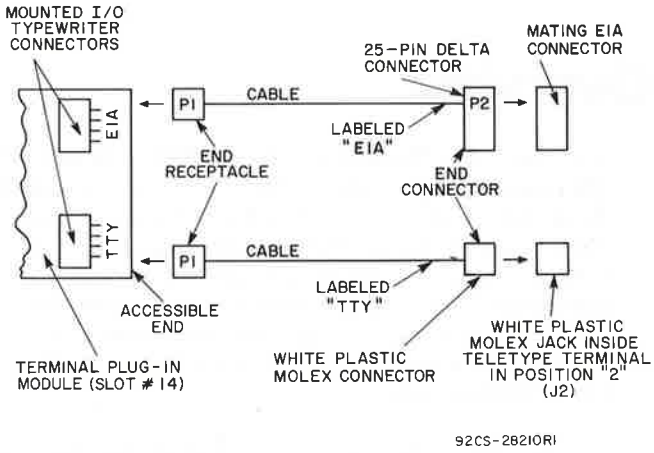


Fig. 1 — Cable connection for I/O data terminals.

The two connectors on the terminal board are labeled appropriately. The connector on the other end of the cable plugs into the terminal. For a Teletype terminal, remove its cover and plug the cable into connector position 2 (J2) in the array of white plastic Molex connectors located in the back of the unit. For an EIA terminal, plug the EIA connector on the cable into the receptacle on the back of the data terminal. Fig. 1 shows these connections.

When the Silent 700 terminal is used, the cable supplied with that device should be connected to the CDS via the EIA cable.

Put the terminal in the Line mode, select the appropriate baud rate and set for full-duplex operation before attempting to use any terminal.

Appendix B gives instructions for converting a TTY from half- to full-duplex operation. The interface is a 20-mA current loop. Make necessary changes per Appendix B to convert a TTY to 20-mA operation. If a TTY is to be used with the assembler program, the remote reader control circuit should be installed in accordance with the instructions in Appendix C. Once installed, its switch should be set in the MANUAL position before continuing.

Install the power cable and switch power on. Press RESET followed by RUNU. This sequence will cause the RUN light to go ON and the system is now operating with UT20 in control.

This program begins by reading the first keyboard input character to define for itself the terminal character rate and whether it should "echo" typed information to the data terminal printer. If the terminal is operating in the full-duplex mode, the user should begin by pressing the RETURN key on the keyboard. For the half-duplex mode, the user should press the LINE FEED key instead. The system then responds with the prompt character (*). It now "knows" the essential characteristics of the I/O data terminal. Reaching this stage verifies that most of the hardware is operating properly.

CDS Checkout Program

Even with little or no knowledge of the COSMAC command repertoire, the user can further verify proper system operation by loading, from the terminal, an elementary test program. For example, a simple time-out loop can be run in which the RUN light goes off after a specific elapsed time from the initiation of execution.

Each line of user keyboard input is terminated by a depression of the RETURN key on the keyboard . The test time-out program can be loaded into memory by typing in

!M0△F8FFB1219191913A0300(CR)

The system will reply with the prompt character. One can verify proper loading by entering

?M0△A(CR)

The system will print the characters just entered (after the memory location addressed, "0000" in this case) and will return the prompt character again. The time-out program can then be run either by entering

$P0(CR)

or by depressing RESET followed by RUN. In either case, the RUN indicator should go off after approximately 2.6 seconds. This step establishes that the read-write memory (RAM) is operational.

● In this Manual, (CR) at the end of an example of a user keyboard input denotes the terminating carriage RETURN. Spaces in an input line will be denoted by blanks in the example or, for additional emphasis, by the symbol △

Loading and Outputting Programs

Programs may be entered manually by use of the !M command, just described. This and other Monitor commands are covered in detail in the next section. Ordinarily, programs will be loaded from paper tape via a TTY, from magnetic tape cassettes via the TI terminal, or from a floppy diskette via the Floppy Disk system. The latter is covered separately in the **RCA COSMAC Floppy Disk System CDP18S805 Instruction Manual MPM-217.** RCA-supplied CDS programs are designed to work in the full-duplex mode.

Following are the methods used with paper tape and magnetic cartridges:

Paper Tape Systems

To load a paper tape:

1) Press RESET, followed by RUN U.

2) Press the RETURN key (CR) on the TTY. Make sure it is in the LINE Mode and the installed switch is in the MANUAL position.

3) UT20 will return the symbol * indicating it is ready to accept commands.

4) Position the tape in the header and turn on the tape recorder.

5) When loading is complete, UT20 will issue another *.

6) Start the program by typing $U0(CR).

If preferred, typing can be suppressed during paper-tape loading by pressing the LINE FEED key instead of CR at step 2. In this case, the user should re-initialize the system after loading by pressing RESET, RUN U, and CR before attempting to start the loaded program.

UT20 monitors the program being loaded and will issue a ? if a format error is detected. If an error is detected, stop and reload the tape from the beginning.

To punch reloadable tape:

1) With the TTY in the LOCAL mode, position tape in the punch, turn the punch ON, and make a header of nulls (control-shift-P).

2) Type !Maaaa △ where aaaa is the hex address of where the data is to be reloaded (normally location 0000).

3) Turn the punch OFF and put the TTY in the LINE mode.

4) Initialize the CDS with a RESET, RUNU, followed by a RETURN (CR).

5) Next, type ?Maaaa △count, where the address is the starting address of data to be read from memory, and count is the number of hex bytes to be punched.

6) Turn the punch ON and press CR. After the tape is punched, some more nulls should be added to its end.

The assembler and editor programs automatically punch reloadable tape as described in the Section titled **CDS Resident Software Development Aids.**

Magnetic Tape Systems

To load a magnetic tape:

1) Press RESET, RUN U, then CR.

2) UT20 will return the symbol *.

3) Mount the cassette. Rewind it and press LOAD/FF to advance to the first record. Make sure the drive is in the LINE and PLAYBACK mode.

4) When loading is completed, UT20 will issue another *. Start the program by typing $U0(CR).

Typing during load can be suppressed by turning the printer OFF. If a ? is typed during loading, an error has been detected and the tape should be reloaded.

To record reloadable tape:

1) With the terminal in the LOCAL and RECORD mode, mount a blank cartridge.

2) Type !Maaaa △ where aaaa is the hex address of where data is to be reloaded (normally location 0000).

3) Switch to the LINE and PLAYBACK mode and initialize the CDS with a RESET, RUN U, and CR.

4) Type ?Maaaa△count, where the address is the starting address of the data to be read from memory and count is the number of hex bytes to be recorded.

5) Turn the Record Control switch ON and press CR. After the data has been output, UT20 will issue another *.

For another system checkout program using the supplied "Deduce" game, refer to Appendix H.

Introduction to the Monitor Software UT20

Utility Commands

The CDP18S005 COSMAC Development System includes a Monitor program, known as UT20, which performs commonly required functions of running the terminal interface, providing a means of reading and generating reloadable tape, giving access to all memory locations, and allows the user to start program at a given location. The following explains in detail the ?M and !M commands already mentioned, plus others not yet discussed.

In general, after the system has been RESET, the user has two choices: pressing RUN begins execution of his program at location 0000, while pressing RUN U begins execution of UT20 (at 8000). After pressing RUN U, the user next presses either a LF (line feed) or a CR (carriage return) key, depending upon his installation. A CR initiates FULL DUPLEX operation, an LF, HALF DUPLEX. Besides establishing the need to echo, UT20 uses this input to calculate the timing parameters necessary to run the terminal. Thus, a single program can operate with wide variations in clock speed or terminal speed.

When UT20 is ready to accept a command, it types out an asterisk (*) as a prompt character.

?M Commands

To interrogate memory, type a command such as

?M2F5 3(CR)

UT20 responds by printing out the contents of memory beginning at location 02F5: three bytes are printed out as two hex digits each. Each line of output begins with the address, and data is grouped in 2-byte (4-digit) blocks. When necessary, new lines are begun every 16 bytes, with the previous lines ending in semicolons. The user may enter any number of digits to specify the beginning location (leading zeroes are implied, if necessary). If more than four digits are entered, only the last four are used. The number of bytes to be typed out should be in hex. Again, if more than four digits are entered, only the last four are used. This feature allows the correction of a mistake simply by continuing the type and terminating the typed sequence with the correct 4-digit values (230024 is, effectively, 0024). If the number of bytes to be typed is not specified, one byte is assumed. For example:

?M2F5(CR)

would result in the typeout of the one byte at location 02F5.

When the user wants to punch a reloadable paper tape, he requests a memory type-out as previously described.

!M Commands

In general, data is entered into memory, by means of a command such as

!M12F 434F534D4143(CR)

This command enters six bytes (two hex digits each) into memory beginning at location 012F. Once again, the starting location is determined by the last four digits entered. Data is entered into memory after each two hex digits are typed. If the user types an odd number of digits, the last digit is ignored, and the error message ('?') is typed out. It is therefore only necessary to re-enter the last byte.

The !M command provides two options that facilitate memory loading. First, a string of data can be extended from line to line by typing in a comma just before the normal CR. (In this case press the CR-LF (carriage return-line feed) keys before beginning a new line.) For example,

```
!M23 56789ABC,(CR) (LF)
DEF0123456,(CR) (LF)
3047(CR)
```

enters 11 successive bytes beginning at location 0023. Between successive hex pairs while data is being entered, any non-hex character except the comma (and semicolon, as will be discussed) is ignored. This arrangement permits arbitrary LF's, spaces (for readability), nulls (generated by the utility program or by a time-share system to give the carriage time to return), etc.

As a second optional form of data entry, a string of input data can be terminated by a semicolon (and a CR). The utility program then expects more data to follow on the next line, but preceded by a new beginning address. The line must have the format of an !M command, but with the !M omitted. This option provides the mechanism for reading in a paper tape previously punched out as a result of the ?M command. (Recall the format of multiline ?M outputs discussed above.)

The utility program ignores all non-hex characters following !M, allowing CR, LF, and nulls to be inserted in the tape without disturbing the !M command. The semicolon feature allows non-contiguous memory areas to be loaded.

$U Commands

The $U command is used to start program execution. For example,

```
$U6C(CR)
```

starts program execution at location 006C with P=X=0. This command will leave the terminal interface and floppy disk interface (if installed) active. Consequently, the user program should not use I/O commands associated with these interfaces. For a further discussion of the terminal interface and the floppy disk interface, see the material on **Module Description and Signal Mnemonics** in the next Section. For further details on the $U command, refer to "Two-Level I/O" under **Input/Output Interfacing** in the next Section.

If only $U(CR) is typed with no address specified, execution will start at location 0000. If more than 4 address digits are typed, only the last 4 will be used.

$P Commands

The $P command is similar to the $U command. For example:

```
$P6C(CR)
```

would also start program execution at location 006C with P=X=0 except, in this case, the terminal and floppy disk interfaces may be disabled. This feature is a convenience for the user so that his program can use I/O commands normally associated with these peripherals.

If no address is specified, program execution starts from location 0000. The function is equivalent to pushing the RESET then RUN P buttons on the control panel. This command also obeys the 'last-4-digits' address rule.

For further details of this command, refer to "Two-Level I/O" under **Input/Output Interfacing** in the next Section.

$L Commands

The $L command is used in systems having a floppy disk. Typing

```
$L
```

causes UT20 to type

```
READ?
```

asking for the unit and track number of the diskette file to be loaded. For a discussion of the disk loader program, refer to the **RCA COSMAC Floppy Disk System II CDP18S805 Instruction Manual** MPM-217. If a floppy disk system is not installed and this command is accidentally activated, simply do a CR after the READ? interrogation. UT20 will type

```
DRIVE NOT ON
```

and issue an *, waiting for another command.

?R Commands

When UT20 is activated (via RESET, RUN U), one of the first things it does is save 13-1/2 of the 16 'R' registers of the CPU in its RAM stack located at address 8C00 for 32 bytes. Registers R0, R1, and R4.1 are altered, but the states of the remaining registers are preserved at the time when UT20 was activated. This feature provides a means of examining most CPU registers for debugging purposes.

The ?R command provides for automatic readback of the stored register states with X's for registers R0, R1, and R4.1 to indicate that they have not been preserved. For example, RESET, RUN U, CR then ?R gives this format:

```
XXXX XXXX 18D4 3821, XX33 B760 8A15 0017
 ↑                                   ↑
R0                                  R7

5518  0717  34AA 8197, A401 6789 A825 01B9
 ↑                                   ↑
R8                                  RF
```

NOTE: the ?R must be the first command given to UT20 after it is started, because UT20 uses the stack itself when other commands are issued. Thus, it may overwrite the preserved registers when executing any command other than ?R.

Summary of UT20 Operating Instructions

In summary, after receiving the prompt character '*' the user may type

?M(address) △(optional count) (CR)

!M(address) △(data) (Optional , or ;) (CR)

(where the data may have non-hex digits between each hex pair)

$P (optional address) (CR)
$U (optional address) (CR)
$L
?R (CR)

UT20 ignores initial characters until it detects ?, !, or $. Then inputs which are not compatible with the above formats cause an error message (?).

A further detailed summary of these basic operating instructions is given below, repeating the information just given in a more concise form.

1. After pressing "RUN U", the user should press CR (for full-duplex operation). This instruction sets up the bit-serial timing and specifies echo or not.

2. UT20 will return * as a prompt.

3. Following *, UT20 ignores all characters until one of ?, $, or ! is typed in.

4. Following ?M or !M, UT20 waits for a hex character. It then assembles an address. If more than four hex digits are typed, only the last four are used. Next, a space is required. Note: △ denotes a space.

   a. For **?M addr**△ a hex count may follow (again, only the last four digits are kept), and the command is terminated by CR. If no count is entered, one byte will be typed.
   b. For **!M addr**△ data must follow. An even number of hex digits is required. Before each hex pair arbitrary filler, except for a CR, comma, or semicolon, is allowed. CR terminates the command, unless it is immediately preceded by a comma or, as is generally the case, by a semicolon.

      i. In case of **comma CR** the user must insert an LF for UT20 to continue to accept data. This procedure is a form of line continuation.
      ii. In case of a **semicolon** all following characters are ignored until the CR is typed. Then, the user must again provide an LF, and UT20 continues as if it had received optional filler, then a starting address, then a space, and then data.
      iii. The **!M command** can be followed by as many continuation lines as needed, mixed between the two types if desired, and is finally terminated with a CR not preceded by a comma or semicolon.

5. Commands **$P** or **$U** may be followed by a starting address. The last 4 digits are used if more than 4 are typed in. If no address is given, 0 is assumed. Program execution begins at the specified location with R0 as the program counter •. The $P command disables the terminal and floppy disk interfaces whereas $U does not.

6. Command **$L** starts the floppy disk loader program which will issue the prompt

READ?

A proper response is a 4-digit number requesting unit and track number, followed by a CR. If an error is detected during the read operation, a diagnostic message is printed.

7. Command **?R** causes a readout of the 16 R registers saved when UT20 is initialized. X's are written for those registers not preserved.

• $P and $U always begin with R0 as program counter. This arrangement is consistent with the fact that P=0 and X=0 after the CPU is RESET. Refer to the CDP1802 data sheet for other actions of RESET.

8. When a !M, ?M or ?R command is accepted and completed, UT20 types another * prompt.

9. When UT20 detects bad syntax, it types out a ? and returns the carriage. If a mistake is made when data is entered (by typing in an odd number of digits), all data will have been entered except the last hex digit. Note that the "only-last-four-digits" rule in the address field allows the user to correct an address error without retyping the whole command. For example, a mistaken 234 can be corrected by continuing. Thus, 2340235 is, effectively, 0235. A bad command can be aborted by typing in any illegal character except after !M or ?M or between input hex data pairs. In these cases, the user should type any digit and then, for example, a period.

Terminal Interfacing

ASCII Coding

The CDS is designed to interface to a data terminal via a serial ASCII code using either a 20-mA current loop or an EIA RS232C standard electrical interface. When a key is struck on a TTY terminal, the information denoting that character is converted to its ASCII code and appears on the output terminals as a serial data-bit stream. The serial data originating at the TTY for the letter 'M' is shown in Fig. 2. The character is framed by a start bit B and

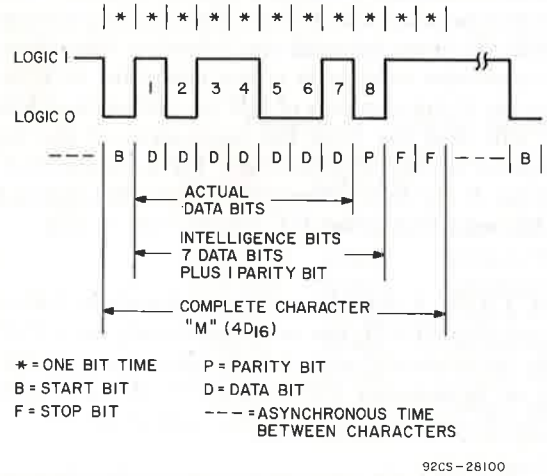


Fig. 2 — Data terminal bit serial output for the character "M".

two stop bits FF. By convention two stop bits are used for data transmission at 10 characters per second although 1, 1½, or 2 are also acceptable outputs from various different terminals. A parity bit P is also shown. For even parity, the parity bit would be a '1' only if the 7 data bits contain an odd number of '1's. Hence, the total number of 1's in the eight intelligence bits is always an even number. Some data terminals may be set up to generate either even or odd parity. UT20 ignores the parity bit, so either even or odd parity is acceptable.

Data from the CDS is generated with the same format; i.e., a start bit, 7 data bits, a parity bit, and two stop bits. Note that the CDS does not generate parity - the parity bit is always a '1' regardless of the data bits. Therefore, terminals interfacing to the CDS should ignore the parity bit.

UT20 Read and Type Routines

The UT20 read and type routines provide the basic software mechanism for communication between the CDS and data terminal. Several different routines are available to facilitate different types of I/O data transfers.

These routines are designed to allow adoption to various terminal speeds and to determine whether or not characters read in should be "echoed" (ie., typed back immediately). For these purposes, a 'sub-subroutine' called DELAY is included which provides the necessary bit timing delays to the read and type routines. DELAY uses RC as its program counter, which must be set-up to point to location 80EF. UT20 does this automatically when it is started. Any user program using a read or type routine must not alter RC, or must restore it to 80EF before calling a read or type routine. Also, the upper half of register RE (RE.1) contains a control constant. The least significant bit specifies echo (0 denotes echo, 1 denotes no echo). For full-duplex operation, then, this bit is a zero. Again, this is automatically set when UT20 is started and the CR or LF characters received.

The remainder of RE.1 constitutes a timing parameter (TP). TP is calculated as follows:

$$TP = 2 \times \left[\frac{\text{interval between two serial bits}}{320 \times (\text{CPU clock period})} \right] \bullet$$

where the fraction is rounded to the nearest integer. For example, because a Teletype Model 33 operates at 10 characters per second and 11 bits per character, for the CDS running from the supplied 2.0-MHz clock,

● The factor of 2 comes from the fact that the input serial waveform is sampled over two successive bit times. The factor of 320 comes from the fact that the time between samples is 20 instruction times, with each instruction taking 16 clock periods.

$$TP = 2 \times \left[\frac{\frac{1\text{ s}}{10\text{ char}} \times \frac{1\text{ char}}{11\text{ bits}}}{320 \times \frac{1\text{ s}}{2.0 \times 10^6}} \right]$$

$$= 2 \times 56.8 \text{ (rounded to 57)}$$

$$= 114_{10} = 72_{16}$$

Because for proper operation TP must be less than 255, there is a bound on the speed of terminals supported at any given clock rate. Faster terminals or slower clocks can be supported to the extent that roundoff errors do not cause bad timing. For example, at 2.0 MHz and 30 10-bit characters per second,

$$TP = 2 \times \left[\frac{\frac{1}{30} \times \frac{1}{10}}{320/2.0 \times 10^6} \right] = 2(20.8) = 42_{10} = 2A_{16}$$

and the round-off error is small (2100 instead of 20.8). On the other hand, at 2.0 MHz with baud rates above 1200, the round-off error would be too high.

The utility program UT20 uses a subroutine "TIMALC" to generate the operating time constant, using the first character typed in by a user. This routine times the intervals between incoming bits to calculate TP and reads one bit to determine whether or not to echo. Specifically, if a CR is entered while TIMALC is running, then echoes will be provided; an LF suppresses echoes. In either case, RE.1 is loaded with the appropriate constant. TIMALC also loads the subroutine pointer for the DELAY routine. The user of TYPE and READ has the option of calling TIMALC or setting up RE.1 and the pointer to the DELAY routine himself. As a convenience to the user, UT20 leaves RE.1 and RC properly adjusted while performing a $P or $U operation and may be used unless they have been altered by the user.

All read and type routines and TIMALC use **R3** as their **program counter**, and **return** to the caller with **SEP R5**. They can be called directly from a program that can use R5 as its program counter, or they may be called through the Standard Call and Return Technique (SCRT) described in the **User Manual for the CDP1802 COSMAC Microprocessor**, MPM-201 in the Section **Programming Techniques** under "Subroutine Techniques". This programming technique is the most general and is recommended.

RE.1 is reserved for the operating constant (control constant 0 or 1 added to the timing parameter TP) discussed above.

One byte of RAM is needed by read and type routines. These routines assume that R2 points to free RAM and M(R(2)) is altered by them. In general, the user can set R2 to any free RAM location. UT20 uses a byte in its dedicated RAM for this purpose.

RF.1 is used in certain cases to pass the byte being read or typed between the calling routine and these subroutines. When READ is exited, it leaves the input byte in RF.1. When TYPE is entered at location 81A4, the byte to be typed is taken from RF.1.

All routines alter RE.0 and RF.0. They also alter D, DF, and X. The READ routine leaves the input byte in D as well as in RF.1, but the byte in D will be destroyed if the Standard Call and Return Technique is used.

When TIMALC exits, R3.1 is left holding A.1 (READ) = A.1 (TYPE) = 81, but R3.0 is meaningless. When READ exits, R3 is ready for entry at READAH (see Table II). When TYPE exits, R3 is ready for entry at TYPE5 (see same table). When DELAY exits, RC is ready for another call to DELAY. When the Standard Call and Return Technique is used, R3 is automatically set up.

The **READ routine** has two entry points - READ and READAH. The former acts as described above and has no other side-effects. The latter operates just as READ does, but with the following side-effect. If the character read in is a hex character (0-9, A-F) then the 16-bit contents of RD are shifted four bits to the left, and the 4-bit hex equivalent of the input character is entered at the right. DF is then set to 1 on exiting. If the input character is not a hex character, RD is not affected, but DF is set to 0 on exiting.

CAUTION: A READ may immediately be followed by another READ, but not immediately by a TYPE. The caller should wait 1.5 bit times first, which he can do by entering TYPE at TYPE5D or by calling DELAY, with a parameter of 7 or greater.

The DELAY subroutine assumes that the calling program counter is R3. It uses the value, n, of the immediate byte at M(R3) to generate a delay equal to

$$(20 + m\,(2n + 6)) \text{ instruction times}$$

where m is time constant in RE.1 (see previous discussion). It then increments R3 past the calling parameter and returns via a SEP R3.

The **TYPE routine** has five different entry points. Three of them simply specify different places to fetch the character from: TYPE types from RF.1, TYPE5 types from M(R5) and increments R5, and TYPE6 types from M(R6) and increments R6. TYPE5D is an entry which provides a 1.5-bit delay before going to TYPE5. The purpose of this delay is to let an immediately preceding echoed READ process to completion before typing. TYPE2 is an entry which results in RF.1 being typed out in hex form as two hex digits. Each 4-bit half is converted to a ASCII hex digit (0-9, A-F) and separately typed out.

Notice that the READ routines are designed to facilitate repeated calls on READAH, while the TYPE routines are designed for repeated calls to TYPE5. In order to output a string of variable data characters following a READ, given the timing restriction mentioned earlier, it is most logical to call TYPE5D first, using an immediate "punctuation" byte (i.e., non-data such as space, null, etc.) to get the required initial delay and to follow either with repeated calls on TYPE (with the output variable data characters picked up from RF.1) or repeated calls on TYPE5 using immediate data characters. This procedure permits a maximum output character rate.

Another routine, **OSTRNG**, can be used to output a string of characters. OSTRNG picks up the character string pointed to by R6 and tests each character for zero. The characters should be already encoded in ASCII. If a zero is found (ASCII 'null'), the program terminates and returns control to the user via a SEP R5. If the character is not a zero, it is typed out to the terminal. The OSTRNG routine includes a delay on the front end so that it may be called at any time - even following a read.

Tables I and II summarize the functions and calling sequences just described.

TABLE I – UT20 REGISTER UTILIZATION

Register Name	Register Number	Function and Comments
PTER	R0	Altered by UT20 while storing registers. R4.1 is similarly altered.
CL	R1	
ST	R2	Pointer to RAM "work" byte. UT20 uses R2 = 8C00.
SUB	R3	Program counter for all routines except DELAY.
PC	R5	Program counter for UT20 which calls the routines above.
DELAY	RC	Program counter for the DELAY routine. Points to DELAY1 in memory.
ASL	RD	Assembled into by READAH (input hex digits).
AUX	RE	RE.1 holds time constant and echo bit. RE.0 is used by all READ and TYPE routines and by TIMALC, OSTRNG, and CKHEX.
CHAR	RF	RF.1 holds input/output ASCII character. RF.0 is used by all READ and TYPE routines and by TIMALC, OSTRNG, and CKHEX.

TABLE II – UT20 READ AND TYPE CALLING SEQUENCE

Entry Name	Absolute Address	
READ	813E	Input ASCII → RF.1, D (if non-standard linkage)
READAH	813B	Same as READ. If hex character, DIGIT → RD (see text)
TYPE5D	819C	1.5-bit delay. Then TYPE5 function.
TYPE5	81A0	Output ASCII character at M(R5). Then increment R5.
TYPE6	81A2	Output ASCII character at M(R6). Then increment R6.
TYPE	81A4	Output ASCII character at RF.1.
TYPE2	81AE	Output hex digit pair in RF.1.
TIMALC	80FE	Read input character and set up control byte in RE.1. Initialize RC to point to DELAY1.
DELAY1	80EF	Delay, as function of M(R3) (see text). Then R3 + 1.
OSTRNG	83F0	Output ASCII string at M(R6). Data byte 00 ends typeout.

Notes

(1) All routines, except DELAY, use R3 as program counter, exit with SEP5, and alter registers X, D, DF, RE, RF and location M(R2).

(2) DELAY routine uses RC as program counter, exits with SEP3 after incrementing R3, and alters registers X, D, DF, and RE.

(3) READ and READAH exit with R3 pointing back at READAH.

(4) All five TYPE routines exit with R3 pointing at TYPE5.

Examples of UT20 Read and Type Usage

The following examples should help clarify how to use the UT20 read and type subroutines. Most examples use the standard subroutine linkage which requires that R2 point at a free RAM location.

Read Routines

This sample program will read four ASCII hex characters into register RD translating them from ASCII to hex in the process. Reading will terminate when a carriage return is entered. Entry of a non-hex digit other than a carriage return will cause a branch to an error program which will type out a "?". This sample program uses the standard subroutine call and return linkage.

```
READAH=#813B

LOOP: SEP R4,A(READAH) ..Call the hex
                       ..read program
      BDF LOOP         ..As long as ASCII hex
                       ..digits are entered
                       ..Read and shift in
                       ..Fall through if not hex
                       ..character
      GHI RF           ..See what character was last
                       ..entered
      XRI #0D          ..Was it carriage return
      BNZ ERROR        ..If not, BR to error
                       ..Characters entered are now
                       ..in RD
```

The READ routine (at 813E) could be used similarly to enter characters; however, READ only enters them one at a time into RF.1 (and D) writing over the previous entry. Note that, even though incoming data is entered into D, the subroutine return program alters D. Therefore, valid data will only be found in RF.1 (and RD when READAH is used) if the standard subroutine call and return programs are used. An alternative technique is to use R5 as the main program counter (since all read and type routines terminate with a SEP R5) and call the program with a SEP R3 (since all read and type routines use R3 as their program counter). The following example illustrates this technique.

Type Routines

EXAMPLE 1: This program outputs a single character using the TYPE5 routine. It uses R5 as the program counter.

```
LDI #81    ..Set R3 to TYPE5 routine
PHI R3
LDI #A0
PLO R3
LDI #FF    ..Set R2 to free RAM location #3FFF
PLO R2
LDI #3F
PHI R2
SEP R3     ..Call type
,T'R'      ..An "R" will be typed
yy         ..Next instruction
```

The TYPE5D routine is used in the same way.

EXAMPLE 2: This program outputs a character using the TYPE6 routine. Note that R6 should be the program counter for the program calling TYPE6 if the character to be typed is an immediate byte because TYPE6 must always be from M(R6). But, because TYPE6 exits with SEP 5, TYPE6 must always be called using standard subroutine linkage for typing an immediate byte. An alternative is to use R5 as the main program counter but point R6 at the memory location containing the byte to be typed. This example uses standard subroutine linkage.

```
SEP R4     ..Branch to the call routine
,#81A2     ..Address of TYPE6
,T'?'      ..Byte to be typed out
yy         ..Next instruction
```

EXAMPLE 3: The TYPE and TYPE2 routines pick up the byte in RF.1 for typing. TYPE simply outputs the character, whereas TYPE2 considers RF.1 a hex digit pair which it encodes in ASCII before typing. This example types out the hex digits 'D5', and uses standard subroutine linkage.

```
LDI #D5    ..Load hex digits D5
PHI RF     ..Into RF.1
SEP 4      ..Call TYPE2
,#81AE
yy         ..Next instruction
```

Note that all type routines, except TYPE2, expect the character they pick up to be already encoded in ASCII.

EXAMPLE 4: An entire message can be typed by using the OSTRNG routine. The ASCII bytes pointed to by R6 will be typed. When a '00' byte is detected, OSTRNG returns to the caller. This example will output the string

```
RCA COSMAC
MICROPROCESSOR
```

The standard call and return linkage is assumed.

```
OSTRNG = #83F0
.
.
.
SEP R4,A(OSTRNG)          ..Call OSTRNG
DC T'RCA COSMAC'          ..1st Line
   ,#0D0A                 ..(CR) (LF)
   ,T'MICROPROCESSOR'..2nd Line
   ,#00                   ..End of Text
.
.
.
```

Additional Utility Routines

ASCII to Hex Conversion Routine

The ASCII to hex conversion, **CKHEX**, examines the ASCII character in RF.1. If this character is not a hex digit, CKHEX returns to the user (via SEP R5) with DF = 0. If the character is hex, CKHEX returns with RE.0 = hex digit, DF = 1 and with the digit shifted into the least significant 4 bits of register RD. CKHEX uses the registers described above and, as with the other routines, is most readily handled via the standard call and return techniques. CKHEX is located at 83FC.

Initialization Routines

Two routines are provided, **INIT1** and **INIT2**, which initialize CPU registers for the standard call and return technique. These routines set up registers as follows:

R2 = R(X) - pointing to the last (highest) available user RAM location (below 8000).
R3 - will become the program counter on return
R4 - pointing to the call routine in UT20
R5 - pointing to the return routine in UT20

The INIT programs examine user memory area (below address 8000) and determine how much memory is present. They set R2 to the highest available RAM address, which is 03FF for the CDS as supplied (with one 4-kilobyte RAM card).

The only difference between INIT1 and INIT2 is the location to which they return. INIT1 returns to location 0005 with P = 3, while INIT2 simply returns by setting P = 3 and assumes that the user has already set R3 pointing to the correct return point. These programs are intended as a convenience to free the user from generating the overhead code required by the standard subroutine technique. They may also be used as an integral part of custom support programs running on the CDS. Their absolute addresses are INIT1 = 83F3 and INIT2 = 83F6. Refer to Appendix G, the UT20 listing, for the absolute addresses of CALL and RET, which will be loaded into R4 and R5 respectively.

Following are examples of the use of these programs:

EXAMPLE 1: Using INIT1
INIT1 = #83F3

```
Address Code Mnemonics  Comment
0000    71   DIS,#00    ..Disable interrupts
0001    00
0002    C0   LBR INIT1  ..Initialize registers
0003    83
0004    F3
0005    --   USRPGM:--  ..User program starts here;
                        ..P = 3
```

EXAMPLE 2: Using INIT2
INIT2 = #83F6

```
Address Code Mnemonics  Comment
0000    71   DIS,#00    ..Disable interrupts
0001    00
0002    F8   LDI A.1    ..Set R3 to return
             (START)
0003    00              ..point
0004    B3   PHI R3
0005    F8   LDI A.0
             (START)
0006    50
0007    A3   PLO R3
0008    C0   LBR INIT2  ..Call INIT2
0009    83
000A    F6
  .
  .
0050    --   START:--   ..User program starts here
                        ..P = 3
```

Routine to Restart UT20

A means is provided to automatically transfer control back to UT20 from a user program. An entry point routine, **GOUT20**, is provided for this purpose. When entered via this routine, UT20 will restart and issue a * prompt to the terminal. A long branch to GOUT20 at location #83F9 will cause this transfer. UT20 depends on the following conditions upon re-entry:

1) RE.1 = terminal timing constant
2) Two-level I/O is enabled

In order to assure the second condition, the user program must be initiated via the $U command. The GOUT20 routine can be called only by a program having R3 as its program counter.

Additional Notes on UT20

UT20 automatically enables group 1 I/O devices, which includes the terminal and floppy disk interfaces, when it is started. User-added I/O devices wired to the same group-select signal are also enabled. For more information on this subject, refer to "Two-Level I/O" under **Input/Output Interfacing** in the next Section, titled **Hardware Structure of the CDS.**

Interrupts are automatically disabled when UT20 is running. They are re-enabled by either the $P or $U command. Because R1 and R2 must be initialized by a user program before interrupts are allowed, UT20 prohibits start-up via these commands if an Interrupt is pending. Instead, it will type INTERRUPT and issue an *. This feature is a convenience to the user to prevent start-up problems if interrupts have not been externally disabled. If custom hardware is installed that may cause interrupts at start-up, the user program should be started via the RUN P switch.

Programming Methods

Machine Language Programming

With an understanding of the structure and operation of the CPU and the material provided thus far, the reader is prepared to begin using the Development System in an elementary way. For example, he can now understand and possibly modify the time-out test program presented earlier in this Manual. However, almost any hexadecimal (machine language) test program will require use of the I/O typewriter. The most basic way to communicate by the teletypewriter, therefore, will be covered next.

To read a character from the I/O teletypewriter, the user program should transfer control to READ• (in UT20). That is, load R3 with 813E and execute a D3 instruction, making sure that R2 is pointing to a free RAM location. After the typed character is read, the utility routine will return by setting P to 5, i.e., by executing the instruction D5 (making it most convenient if the program counter of the calling routine were 5 to begin with). The ASCII code for the input character (with a 0 parity bit) will be in both RF.1 and in D. The memory location pointed to by R2 and registers RE, RF, X, and DF will have been changed in value (not preserved over the call).

• A list of key UT20 symbolic locations and their corresponding absolute memory addresses is given in Table II.

Because the READ routine uses R3 as its program counter, it is most convenient to branch to READ by a D3 instruction. When READ returns to the caller, R3.0 will contain a modified value, necessitating another initialization if a repeated I/O is to be performed. Because the READ routine uses the values in registers RC and RE which UT20 will normally initialize, it is essential that the user refrain from using these registers unless their values are saved and later restored by his program.

To cause a character to be typed out by the I/O typewriter, the user program should transfer control to TYPE5D at location 819C, by means of a D3 instruction, again making sure that R2 is pointing to a free RAM location. As discussed above, the calling P value should be 5 and, for this case, the ASCII code for the output character should be an immediate byte (i.e., the byte after the D3 instruction). After typing the character, READ will have advance R5 past the argument byte and again return by a D5 execution. M(R(2)), as well as registers RE, RF, X, D, DF, and R3.0 return altered. All other register values are preserved. For the reasons previously cited, the user should again refrain from using registers RC and RE.

Given the ability to execute simple I/O terminal functions, the user can now code elementary test programs to further exercise the COSMAC Development System. As a simple example, consider the routine shown in Fig. 3 that reads two bytes, compares them, and outputs the "larger" of the two.

MEMORY LOCATION	HEX CODE STORED	COMMENTS
00	90	00 → D. (Assumes R0.1 = 0).
01	B2 B5 B6	Clear upper half of pointers.
04	F8 FF A2	M(00FF) is the free RAM location.
07	F8 81 B3	Initialize upper half of I/O call program counter.
0A	F8 3E A7	Lower half of READ address saved in R7.0.
0D	F8 9C B7	Lower half of TYPE address saved in R7.1.
10	F8 27 A6	R6 now points to 0027 (the immediate TYPE byte).
13	F8 17 A5	R5 initialized to 0017 is ready to be the program counter.
16	D5	Change P from 0 to 5.
17	87 A3	R3 now points to READ routine (813E).
19	D3	Call READ. Input character to D and RF.1.
1A	56	Save first character in immediate byte location.
1B	87 A3 D3	READ second character.
1E	E6	X now points to first character.
1F	F7	D − M(R(X)) → D. Subtract first character from second.
20	3B 24	Print the first character if it is the largest.
22	9F 56	Second character moved to immediate byte location.
24	97 A3	R3 now points to TYPE5D (at 819C).
26	D3	Call TYPE. Output byte at the next location (0027).
27	00	Immediate byte storage for TYPE routine.
28	30 17	Loop for another pass.

Fig. 3 – Example of elementary hexadecimal program.

The routine given exhibits register usage compatible with the UT20 READ and TYPE calling sequences. Further, the I/O instructions consist of calls to the appropriate teletypewriter interface routines.

The initialization part of the program is above the dashed line. The main program loop begins at location 17. Each call to the utility program (two READ's and one TYPE) is made by a D3 execution after first initializing R3.0 with the proper half of R7 in which the two lower half address constants (9C and 3E) are stored. R3.1 continues to hold its initialization value of 81. Two characters are read and a subtraction is executed. The resulting immediate TYPE byte is conditional on the results of the subtraction. The free RAM location used by READ and TYPE is 00FF in this case since R2=00FF.

The initialization part of the program is executed with P = 0. R5 is the program counter once the main program loop is entered at location 17. R6 is initialized to point to the immediate TYPE argument location (0027). Notice that X is set to the value 6 (see instruction at location 1E) after the READ routine since READ changes the value of X.

An example of loading and running a program by means of the keyboard is a CDS "session" using the elementary program just discussed. In what follows, underlined text represents UT20 printout; text not underlined represents user input; bracketed text is commentary.

Begin by turning power ON and pressing RESET, followed by RUNU. The RUN light will go on. Then ..

```
(CR)  [Carriage Return to establish timing
       constant and echo]
*     [UT20 prompt character]

!M0 90 B2 B5 B6 F8 FF A2 F8, (CR) (LF)

   81 B3 F8 3E A7 F8 9C B7, (CR) (LF)

   F8 27 A6 F8 17 A5 D5 87, (CR) (LF)

   A3 D3 56 87 A3 D3 E6 F7, (CR) (LF)

   3B 24 9F 56 97 A3 D3 00, (CR) (LF)

   30 17 (CR)

   [The program has now been loaded]
*  [UT20 ready for the next command]
$U(CR)[Begin program execution]
   ABBBAB122212MNN [..etc..]
```

Each character triplet represents one pass through the main program loop consisting of two user input characters and one output character.

The reader may wish to code his own program at this point to verify his knowledge of the CPU instruction set and the read and type routines.

Programming Interface to CSDP

Machine language coding, even of a trivial program, should convince the novice programmer that to do any serious programming, one should take advantage of the set of software support aids available. Veterans in the programming community are already aware of the fundamental necessity for assembly and simulation facilities. Support services are available either by timesharing, i.e., using a system of RCA-developed programs hereafter referred to as CSDP, or by this Development System itself. The user manual for the former set of programs is the **Timesharing Manual for the CDP1802 COSMAC Microprocessor**, MPM-202. For the latter set of programs it is this manual. To do any non-trivial programming, it is essential that the reader be familiar with the facilities provided by these software support systems. If the reader is **not** using CSDP, he should skip this section.

As discussed in the **Timesharing Manual for the RCA CDP1802 COSMAC Microprocessor**, MPM-202, much of program development by CSDP is accomplished without direct CDS involvement. Typically, a source file is constructed, assembled, edited (if the assembler objects to the source code), reassembled,...,etc. The simulator is used to run the program, during which time program bugs are isolated and removed by further editing and reassembly of the source file. Eventually, the object code is ready for loading and running in the real hardware (the COSMAC Development System) for further testing. It is this part of the process that is of concern here.

Already discussed has been the use of the '!M' utility program to load the CDS RAM from the keyboard or from tape. One ultimate purpose of the CSDP system is to generate an object code file, compatible with the required !M format, and (on command) to transmit this file over the telephone link to the CDS system. Clearly, it is possible for the user to write this file onto a tape and subsequently load the CDS using this medium. Of concern here, however, is the automatic mechanism by which the !M-compatible object file coming over the telephone line is loaded into the CDS RAM directly.

There are three different data communication paths. First is the I/O teletypewriter-CDS path, already discussed. Second is the I/O teletypewriter-Timesharing System link, via an appropriate modem, which is implied in the use of CSDP for assembly and simulation. And third is the Timesharing System-CDS link (again via a modem) which is essential to the automatic, direct-load process.

Some switching mechanism is implied by which the Development System serial "Terminal" input signal can come either from the keyboard/tape reader or from the modem carrying data generated by CSDP.

A teletypewriter unit, for instance, requires an external modem (e.g., an acoustic coupler or a data set), and an added external "switchbox" to mechanize the various TTY-CDS, TTY-modem, and modem-CDS paths. It should be an appropriately wired three-position switch. In the "TTY CDS" position the terminal acts as the I/O device for the CDS. In the "TTY-TIMESHARING" position, it acts as normal timesharing terminal. In the "TIMESHARING-LOAD" position, the link is established to allow data from the timesharing system to be automatically loaded into the CDS memory.

If it is assumed that the user has been using the CSDP control program and that an object code file, previously assembled by CSDP, is ready for transmission, the steps required to effect an automatic load of the CDS RAM follow.

Because CSDP will transmit an !M-compatible object file on command from the terminal, it is necessary to properly initialize the utility program so that it is ready for this input. This initialization is done by temporarily switching the terminal to activate the TTY-CDS path only, and pressing RESET, followed by RUNU, followed by the keyboard echo-timing control character LF. Initialization will be followed by a return of the prompt character indicating that UT20 is ready. It will then ignore all subsequent inputs until a ! or ? or $ is detected. It is essential that this local initialization be done at a time **before** the final carriage RETURN, which terminates the "transmit" command to CSDP, yet **after** the occurrence of any characters in this command string recognizable by UT20. Thus, the final "transmit object file" command to CSDP is begun in the TTY-TIMESHARING mode. At the proper point, UT20 initialization occurs, as discussed above. Then, the terminal is switched back to TIMESHARING LOAD and the command is completed. All subsequent characters are ignored by UT20 until it receives the loading !M indicating the beginning of object file transmission.

CSDP indicates that it is ready for a user command when it outputs to the terminal the prompt characters DBG. Assuming that the assembled file is ready for transmission, the following two alternative CSDP commands will effect the transmission:

1. $X△ *File Name* △ *Start RAM Location* △ *End RAM Location* where *File Name* is the name of the file or the **device** which will

receive the specified contents of CSDP's simulator memory. These contents are normally object-code generated as the result of a just-completed assembly. If *File Name* is specified as TTY, the object code will be transmitted over the telephone line to the terminal. For example,

$X TTY #0 #1FF (CR)

will result in the transmission of the comma-continuation form of the !M object file, loading the lowest part (200_{16} bytes) of the CDS RAM. (Recall that the utility program must be properly initialized just prior to the final CR.) The !M form is more compact and "relocatable".

2. $Y△ *File Name* △ *Start RAM LOC* △ *End RAM Loc* where the same comments above apply to *File Name*. For example,

$Y TTY #20 #150 (CR)

will result in the transmission of the semicolon-continuation form of the !M file (more readable, since each line begins with an address value). This form is particularly useful for "scatter loading" of subroutines or other memory patches.

If the CDS has been initialized properly, just before the transmission begins, the object file will be loaded into RAM automatically. The user can then proceed to run the program with the I/O data terminal in TTY-CDS mode, using standard CDS facilities, i.e., $P or $U by means of the utility program or a RESET, RUN P sequence.

An example follows of a CSDP-Development System session using a Level-1 assembly language version of the sample program given in Fig. 3. The source program is listed in Fig. 4. For this example, the program is written to begin at memory location 0001 to illustrate use of the ORG statement. The comments in the listing should be sufficient to permit the reader to establish correspondence with the detailed hex code in Fig. 3. Assume that this source program is entered from the keyboard into the timesharing system, as a file named SAMPLE. Once this entry is made, the user calls for activation of the CSDP program. When CSDP is ready for a command, it prompts the terminal with the DBG message. By entering the CSDP command $A SAMPLE, TTY (CR), the user calls for an assembly of this file with listing and diagnostics printed to the terminal. The output received is indicated in Fig. 5●. The code listed there (the same as that in the hex program generated earlier) has now been loaded into the CSDP simulator's memory. After generating the listing, CSDP again returns with the DBG prompt.

At this point, the user follows the instructions given earlier, that is, input of the $X or $Y command with appropriate UT20 initialization, to effect an automatic load of the object file into the CDS RAM. The sequence of steps is as follows:

1. Begin the transmit command to CSDP by typing $α TTY #0 #30 (where "α" is either X or Y), but without typing the final carriage RETURN yet.
2. Switch the terminal to the TTY-CDS mode.
3. Press RESET, RUNU, CR or LF (for full or half duplex), and receive the * prompt as an answer from the utility program.
4. Switch back to TTY-MODEM mode. Switch the selector switch to the "CSDP LOAD" position at this point.
5. Press the final carriage RETURN, terminating the CSDP transmit command. For this case, 30_{16} bytes will be transmitted.

While the object file is being transmitted to the Development System, it is also printed on the data terminal. For the example given, the CSDP command $X TTY #0 #30 will result in

```
!M0000

FF90B2B5B6F8FFA2F881B3F83EA7F89CB7F828A6,

F818A5D587A3D35687A3D3E6F73B259F5697A3D3,

FF3018FFFFFFFFFFFF

DBG
```

printed on the terminal during the loading of RAM. On the other hand, for the same example, the CSDP command $Y TTY #0 #30 will result in

● Refer to the **Timesharing Manual for the RCA CDP1802 COSMAC Microprocessor**, MPM-202, for detailed CSDP user instructions and an explanation of the assembly listing format. Note that for forward references, the code displayed in the listing does not correspond exactly to the contents of the simulated memory.

```
!M

0000 FF90 B2B5 B6F8 FFA2 F881 B3F8 3EA7 F89C;

0010 B7F8 28A6 F818 A5D5 87A3 D356 87A3 D3E6;

0020 F73B 259F 5697 A3D3 0030 18FF FFFF FFFF;

0030 FF

DBG
```

printed on the terminal during the loading process. Notice that, after completion of transmission, CSDP comes back in its command mode with the DBG prompt.

The user may now log off from the timesharing system after a "$Q" to CSDP and then switch the data terminal to the TTY-CDS mode to verify loading, via the ?M command, and to run the program.

```
RDPTR = #3E          . . LOWER HALF READ ENTRY
WRPTR = #9C          . . LOWER HALF TYPE5D ENTRY
IO = R3              . . UT3 READ-TYPE PROGRAM COUNTER
PC = R5              . . CALLING PROGRAM COUNTER
ARG = R6             . . POINTER TO IMMEDIATE TYPE BYTE
IOPTR = R7           . . HOLDS WRPTR AND RDPTR VALUES
ORG #01              . . PROGRAM STARTS AT LOC 1
. . . . . INITIALIZATION PHASE BEGINS HERE . . . . . .
GHI R0               . . CLEAR D, SINCE R0.1 IS ZERO
PHI R2
PLO R2               . . R2 POINTS AT FREE LOC (ZERO)
PHI PC
PHI ARG              . . CLEAR UPPER HALVES OF LOCAL PTRS
LDI #81
PHI IO               . . INITIALIZE UPPER HALF UT3 PC
LDI RDPTR
PLO IOPTR
LDI WRPTR
PHI IOPTR            . . LOWER HALVES OF UT3 ENTRIES
LDI A. 0 (ARG1)
PLO ARG              . . POINTER TO TYPE SAVE BYTE
LDI A. 0 (LOOP)
PLO PC               . . LOCAL PROGRAM COUNTER READY
SEP PC               . . SWITCH PROGRAM COUNTERS
. . . . . MAIN PROGRAM LOOP BEGINS HERE . . . . . .
LOOP: GLO IOPTR
PLO IO               . . POINT TO READ
SEP IO               . . CALL READ. FIRST CHAR TO D
STR ARG              . . SAVE IT
GLO IOPTR
PLO IO               . . REPOINT TO READ
SEP IO               . . CALL READ. SECOND CHAR TO D, RF . 1
SEX ARG              . . RESTORE ARG POINTER
SM                   . . SECOND CHAR MINUS FIRST
BM TYPE              . . EXIT IF FIRST CHAR IS LARGER
GHI RF
STR ARG              . . SECOND CHAR TO OUTPUT ARG LOC
TYPE: GHI IOPTR
PLO IO               . . POINT TO TYPE5D
SEP IO               . . CALL TYPE.
ARG1:  ORG*+#01      . . IMMEDIATE BYTE ARG AND SAVE LOC
BR LOOP              . . LOOP FOR ANOTHER PASS
END
```

Fig. 4 – Source code of sample program for CSDP timesharing assembler.

```
DBG $A SAMPLE, TTY

FL   LOC    COSMAC CODE   LNNO  SOURCE LINE
     0000                    1  RDPTR = #3E       .. LOWER HALF READ ENTRY
     0000                    2  WRPTR = #9C       . . LOWER HALF TYPE5D ENTRY
     0000                    3  IO = R3           . . UT3 READ-TYPE PROGRAM COUNTER
     0000                    4  PC = R5           . . CALLING PROGRAM COUNTER
     0000                    5  ARG = R6          . . POINTER TO IMMEDIATE TYPE BYTE
     0000                    6  IOPTR = R7        . . HOLDS WRPTR AND RDPTR VALUES
     0001                    7  ORG #01           . . PROGRAM STARTS AT LOC 1
     0001                    8  . . . . . INITIALIZATION PHASE BEGINS HERE . . . . . .
     0001 90                 9  GHI  R0           . . CLEAR D, SINCE R0.1 IS ZERO
     0002 B2                10  PHI R2
     0003 A2                11  PLO R2           . . R2 POINTS AT FREE LOC (ZERO)
     0004 B5                12  PHI PC
     0005 B6                13  PHI ARG          . . CLEAR UPPER HALVES OF LOCAL PTRS
     0006 F881              14  LDI #81
     0008 B3                15  PHI IO           . . INITIALIZE UPPER HALF UT3 PC
     0009 F83E              16  LDI RDPTR
     000B A7                17  PLO IOPTR
     000C F89C              18  LDI WRPTR
     000E B7                19  PHI IOPTR        . . LOWER HALVES OF UT3 ENTRIES
F    000F F800              20  LDI A.0 (ARG1)
     0011 A6                21  PLO ARG          . . POINTER TO TYPE SAVE BYTE
F    0012 F800              22  LDI A.0 (LOOP)
     0014 A5                23  PLO PC           . . LOCAL PROGRAM COUNTER READY
     0015 D5                24  SEP PC           . . SWITCH PROGRAM COUNTERS
     0016                   25  . . . . . MAIN PROGRAM LOOP BEGINS HERE . . . . . .
     0016 87                26  LOOP: GLO IOPTR
     0017 A3                27  PLO IO           . . POINT TO READ
     0018 D3                28  SEP IO           . . CALL READ. FIRST CHAR TO D
     0019 56                29  STR ARG          . . SAVE IT
     001A 87                30  GLO IOPTR
     001B A3                31  PLO IO           . . REPOINT TO READ
     001C D3                32  SEP IO           . . CALL READ. SECOND CHAR TO D, RF.1
     001D E6                33  SEX ARG          . . RESTORE ARG POINTER
     001E F7                34  SM               . . SECOND CHAR MINUS FIRST
F    001F 3B00              35  BM TYPE          . . EXIT IF FIRST CHAR IS LARGER
     0021 9F                36  GHI RF
     0022 56                37  STR ARG          . . SECOND CHAR TO OUTPUT ARG LOC
     0023 97                38  TYPE: GHI IOPTR
     0024 A3                39  PLO IO           . . POINT TO TYPE5D
     0025 D3                40  SEP IO           . . CALL TYPE.
     0027                   41  ARG1: ORG*+#01   . . IMMEDIATE BYTE ARG AND SAVE LOC
     0027 3016              42  BR LOOP          . . LOOP FOR ANOTHER PASS
     0029                   43  END
NO   UNDEFINED LABELS
NO   UNDEFINED SYMBOLICS
       DBG
```

Fig. 5 — CSDP assembly listing for sample program.

Hardware Structure of the CDS

This section of the manual is organized to a present a "top down" explanation of the hardware structure of the COSMAC Development System. First, an over-all system block diagram is given containing sufficient detail to explain the basic functions of each of the modules and to indicate all of the essential data and control paths in the system. A few "second-order" signals are omitted at this stage for simplicity. This overall diagram is followed by a block diagram of each of the modules. These diagrams are designed to provide sufficient detail (signal mnemonics, timing information, etc.) so that further user analysis of the individual module logic diagrams (given in Appendix D) should be unnecessary. The assignment of connector pins to signals is omitted at this point. This detail is found in the backplane wiring schedule in Appendix A.

A fundamental prerequisite to understanding the structure of the CDS is a detailed familiarity with the COSMAC CPU interface and instruction set, as described in the **User Manual for the RCA CD-P1802 COSMAC Microprocessor**, MPM-201.

System Block Diagram

The broad organization of the Development System is indicated in Fig. 6. The CPU module interfaces with an I/O system to its right in the diagram and with a memory system to its left. The switches and indicators on the Control Panel (at the top of the diagram) communicate with the system through the CLOCK and CONTROL Module, which also contains the system clock.

Most of the signal paths in the diagram are labeled with signal mnemonics in parentheses. The notation (i:j) is used to denote a parallel set of signals, each labeled with a unique index in the range i to j. Thus, for example, A(14:12) represents the bundle of three parallel signals A14, A13 and A12. Note further that where additional emphasis is required to distinguish it from the letter "O", a "Ø" is used for the numeral zero.

The superscript after each module name in the diagram denotes the plug-in connector position or slot number in the CDS next to which the module is assigned.

Physically, the Development System consists of a card nest with 25 sockets for logic cards interconnected by means of a printed circuit backplane, a power supply, a chassis which will mount via a 19" rack or a cabinet, and a hinged panel on which are mounted control switches and indicators.

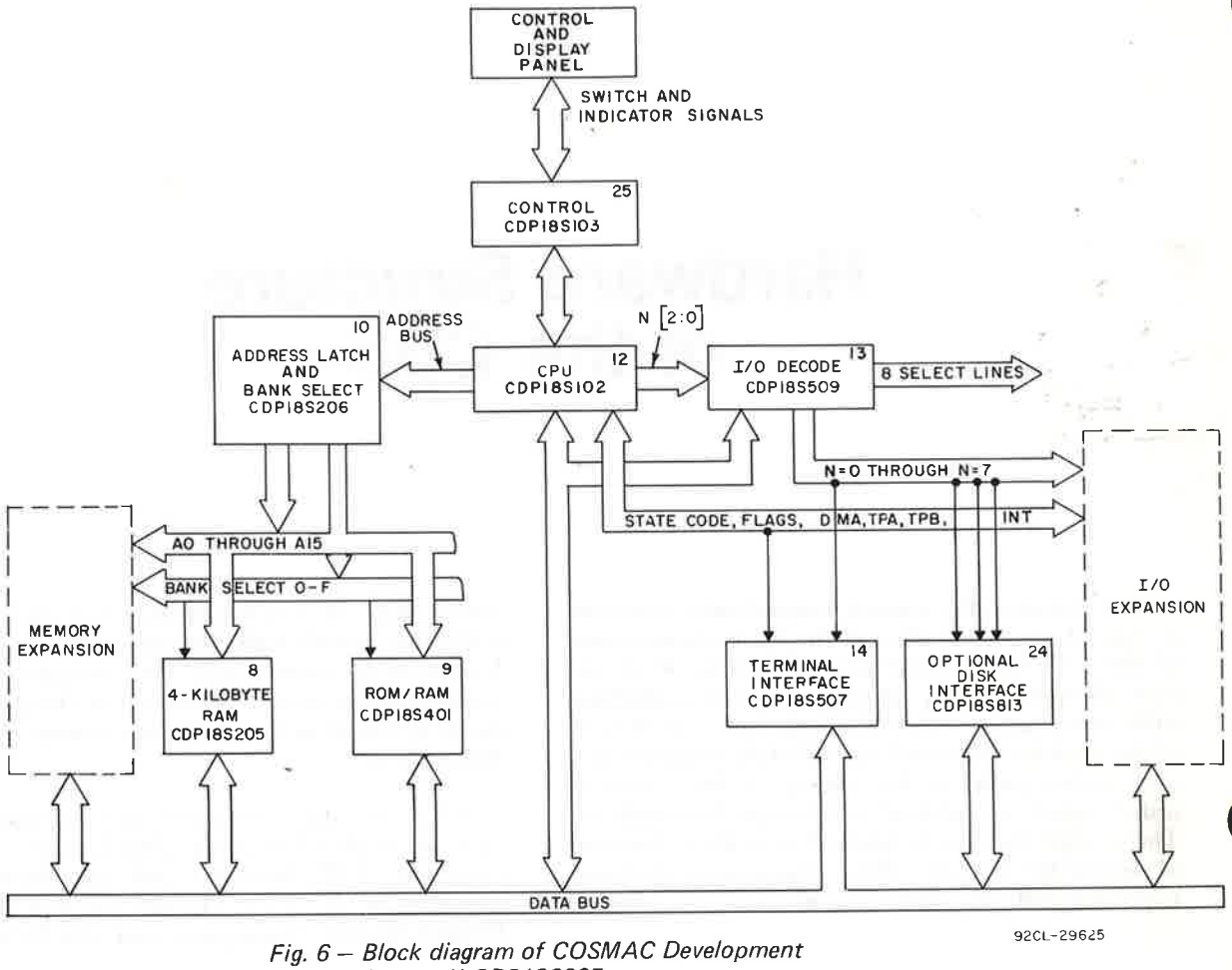


Fig. 6 – Block diagram of COSMAC Development System II CDP18S005.

The control and Display Panel has a PC board on which are mounted the switches, displays, and associated electronics. Via a cable, it communicates with the control module which is inserted into the card nest. The Control module contains circuits to debounce the switches, interface then to the CPU, and perform some logic control and timing. The Control module communicates with the CPU module via wires printed on the backplane.

The CPU module consists of the CDP1802 with a crystal for the on-chip oscillator and electrical buffers for the Data and Address Buses. The CPU module is central to the system, communicating with the I/O and Memory systems to the right and left respectively in the diagram.

On the memory side, the Address Latch and Bank Select module, as its name implies, latches the high-order Memory Address Byte and decodes it into 16 unique Bank Select lines, for use by the entire memory system. On the I/O side, the I/O Decode module accepts the N lines from the CPU, and decodes them into 7 unique lines, N = 1 through N = 7 for use by all I/O controllers. In addition, it latches the Data Bus on a $(61)_{16}$ instruction, and provides eight Select lines for use by all I/O controllers as two-level I/O addressing.

Two memory modules are provided, the ROM/RAM and the 4-kilobyte RAM. The ROM/RAM module holds the system's Utility program ROM (UT20) plus a small (32-byte) RAM for use by that Utility. The 4-kilobyte RAM module provides 4 kilobytes of static RAM storage. Additional 4-kilobyte RAM modifications may be added pre-wired expansion slots.

A Terminal Interface Module is provided, which may be used to communicate with terminals having either the EIA or 20-mA current-loop interface.

All memory and I/O controller modules attach to the common DATA BUS, which is bi-directional. Therefore, these modules must provide a means for disconnecting from the bus, when not using the bus, such as tri-state buffers or transmission gates.

The various control, flags, etc. are distributed to the I/O and memory expansion locations on the backplane. Module select lines, BANK SELECT for memory and SELECT for I/O, are not wired on the backplane PC board, but are available for wire-wrap connections as determined by user system configurations.

Module Description and Signal Mnemonics

Each standard module will be described using a simplified logic or Block diagram. Detailed logic diagrams may be found in Appendix D.

Signal naming conventions are as follows:

The signal name is followed by a hyphen and either the letter N or P. The suffix N means the signal is asserted (true) when that wire is at ground. The suffix P means the signal is asserted (true) when that wire is at the highest logic level (+5 V). Thus the signal name gives the meaning assigned to that conductor, and the suffix defines the electrical value of the asserted (true) state.

A bundle of parallel signals is indicated by a (i:j) notation in the signal name denoting a running index over the range i to j and by the number of parallel signals (in parentheses) labeling the signal path. Inputs which are pulled high or low on the module are indicated with resistor symbols to V_{DD} or to GND. If such an input is not used (not connected), it assumes the high/low level defined by its "pull up/down" resistor. Output signals which are derived from CMOS transmission gates are labeled with a "(T)" in the diagrams. Such outputs may be bussed ("wire-OR'ed") together - assuming only one transmission gate is enabled at a time. An output derived from a transmission gate may also be pulled high or low with a resistor on the board.

Card Nest and Backplane

The backplane of the Card Nest is a double-sided PC board mounted on the back of the CDS. It interconnects the pins of the first 25 of the 33 available connector positions. Some of these positions are occupied with supplied modules, as indicated in Table III. The card positions, viewed from the back, are numbered from left to right.

TABLE III – MODULE POSITION ASSIGNMENTS IN NEST

Connector Position Number	Module	Part Number
1	Memory Bus	
2	Memory Bus	
3	Memory Bus	
4	Memory Bus	
5	Memory Bus	
6	Memory Bus	
7	Memory Bus	
8	4-Kilobyte RAM	CDP18S205
9	ROM/RAM	CDP18S401
10	Address Latch and Memory Bank Select	CDP18S206
11	Blank	
12	CPU	CDP18S102
13	I/O Decode	CDP18S509
14	Terminal Interface	CDP18S507
15	I/O Bus	
16	I/O Bus	
17	I/O Bus	
18	I/O Bus	
19	I/O Bus	
20	I/O Bus	
21	I/O Bus	
22	I/O Bus	
23	I/O Bus	
24	(Floppy Disk Interface)	CDP18S813
25	Control	CDP18S103
26 - 32	Power Supply	

The backplane wiring schedule in Appendix A indicates the following types of connection: unused pins, pins interconnected by printed wiring, pins interconnected by a wire-wrap, and pins not connected on the backplane but which have meaning defined by the plugged-in module. A dash indicates an unused pin, In all cases, identically named signals are interconnected. The backplane is laid out so that

signal flow is horizontal, i.e., wherever possible the printed wiring connects identical pins on different connectors. Note that almost all of the printed signal interconnections are made on the exposed side of the backplane. If the user wishes to modify the backplane wiring for some reason, he can scratch out undesired connections and wrap his own.

One notational inconvenience between the CDS documentation and the data sheet for the CDP1802 microprocessor should be explained. In the CDS, the power supply voltage V_{DD} (+5 volts) is the standard high signal level. This voltage is connected on the CPU plug-in board to the microprocessor pin (16) labeled "V_{CC}" on the CDP1802 data sheet. The label "V_{CC}" is not used in CDS documentation. Another microprocessor pin (40) may be connected to a higher power supply voltage level to achieve higher speed, if desired. In the CDS, this pin is connected to the power supply voltage labeled "CPUPWR" (whose level may be 5 volts or higher). The CDP1802 data sheet labels this pin as "V_{DD}". For a CDS for which both positive CPU supply voltages are at 5 volts, the user should not be confused by these alternative notations. A ground strap is provided connecting logic ground to chassis. It may be removed by the user if he has an alternate safety ground in his system.

Each of the solid boxes in the CDS Block Diagram of Fig. 6 corresponds to a supplied module. Both the Block Diagram and the backplane wiring schedule show the modules generally left to right as they appear when viewed from the backplane side. From the front, memory is on the right and the I/O side of the system is on the left.

Some precaution should be exercised in removing and inserting modules into the CDS nest. The module cards are keyed so that they cannot be inserted in improper positions or with improper orientation. However, it is possible for a key to be pulled out by a card removal. When removing a card, care should be taken to exert a lateral force, without twisting the card unnecessarily. It is also possible for a connector contact to be dislodged as a result of improper card removal. A short across to an adjacent contact (1 to A or A to B, for example) can then occur. If trouble develops after a card removal and later reinsertion, careful inspection of the connector involved is advisable before attributing the problem to failed electronics.

Special care should be taken when cards are inserted into or removed from the extender card socket because the socket has no key to limit harmful up-and-down motion or improper card slot insertion.

CPU Module CDP18S102

The CPU is the heart of the COSMAC Development System. It controls and addresses memory, multiplexing the sixteen-bit memory address over a one-byte memory address bus. It manages a bidirectional one-byte data bus. It senses and reacts to external signals - interrupt, DMA input request, DMA output request, and four external flags. It transmits two timing pulses, or syncs, and an encoded CPU state. When executing an input/output instruction (I = 6), it transmits the three-bit "N" field of the instruction. Refer to the **User Manual for the CDP1802 COSMAC Microprocessor, MPM-201**, for details of the CPU operation.

Fig. 7 shows the basic logic contained on this circuit module. All named signals are brought to the backplane connector for use by the system.

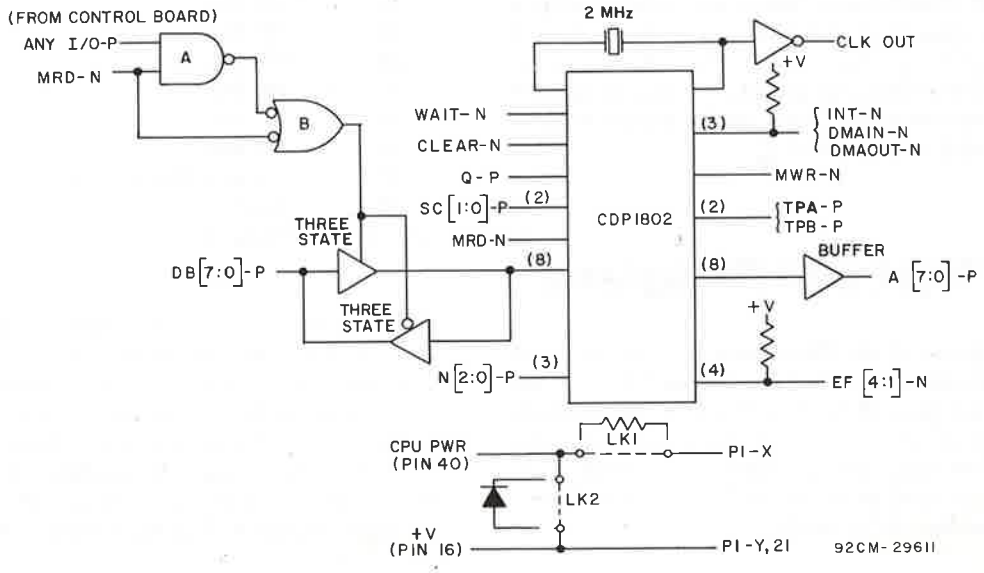


Fig. 7 — CPU Module CDP18S102 block diagram.

The internal oscillator on the CDP1802 is utilized with a 2-MHz crystal. This oscillator provides the internal clocks used by the microprocessor. A buffered clock output is provided for use by any user-developed module requiring a controlled clock. The user may remove the crystal, then provide an external clock by driving pin 12 of this module with an appropriate clock signal.

The Data Bus is buffered by a set of tri-state driver circuits as shown. The drivers on each line are connected head-to-tail, and controlled as to direction. When MRD-N is true, the output of gate B is high, enabling the inbound tri-state driver to transmit. Because outbound drivers require a low level to transmit, they are in a high-impedance state (disconnect) when MRD is true. When MRD is false, the opposite direction is enabled unless an I/O operation is in progress, in which case gate A turns on the inbound devices. The reason for the inbound path during an I/O operation is so that when an input instruction is executed, the data byte may be stored in the D Register of the CPU as well as in memory.

The memory address lines are buffered and sent to the backplane as A(7:0)-P.

The incoming signals EF1-N,EF2-N,EF3-N,EF4-N,INT-N,DMAIN-N, and DMAOUT-N, being negative signals, have pull-up resistors so that when no connection is made, the inputs are logically false. Users of these lines should provide tri-state drivers or transmission gates to pull them to ground so that more than one device may use each line in a wired-or manner.

Power to the CDP1802 is connected to two pins: pin 16 (V_{CC}) powers the chip interface circuits and pin 40 (V_{DD}) powers the internal circuits. These pins are connected to backplane connector pins Y and 21 and pin X respectively and connected together via link LK2. To operate with a higher voltage on V_{DD} in order to gain speed, cut LK2 and provide a higher voltage on pin X. Consult the CDP1802 data sheet for maximum voltage ratings.

It is necessary that, at all times, $V_{CC} \leq V_{DD}$. For this reason, it is recommended that a diode replace LK2 as shown in Fig. 7, and that a current-limiting resistor (approximately 1 kilohm) replace LK1. These additions will prevent possible damage to the device from raising $+V_{DD}$ without a voltage on CPUPWR. The resistor is to limit the current that $+V_{DD}$ may drive into Pin X under those circumstances.

Control Module CDP18S103

The Control Module provides the interface between the control/display panel and the system logic. See Fig. 8.

The control panel switch contacts are brought to the control module via connector J1. These lines are interfaced with appropriate de-bounce or pull-up circuits and logic to control the CPU mode.

The WAIT-N and CLEAR-N inputs to the CDP1802 are controlled to produce one of the four control modes: RESET, RUN, LOAD, or PAUSE.

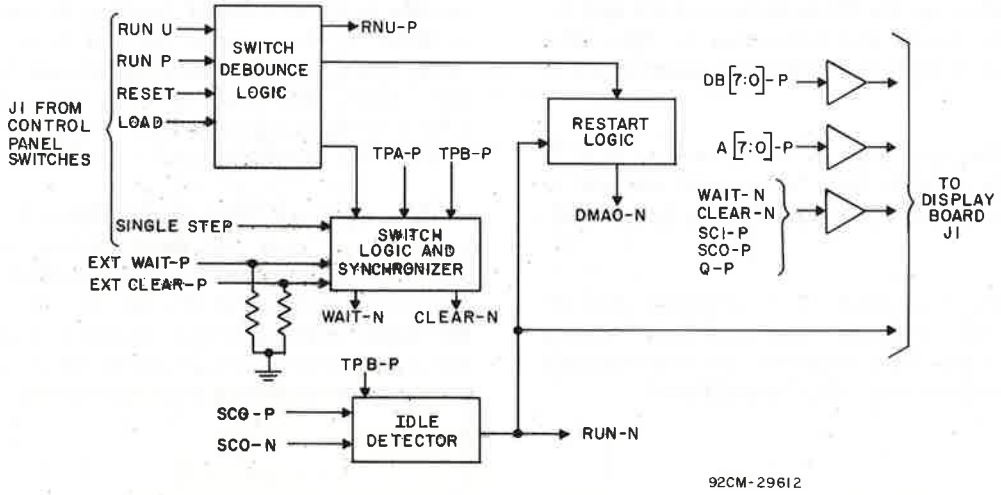


Fig. 8 – Control Module CDP18S103 block diagram.

Depression of the LOAD switch causes both WAIT-N and CLEAR-N to be asserted, defining the LOAD mode, in which data may be loaded into memory using DMA-IN.

Depression of the RESET switch results in CLEAR-N asserted and WAIT-N false which puts the CPU into the RESET mode.

Depression of RUN P or RUN U results in both WAIT-N and CLEAR-N false, putting the CPU into the RUN mode. In the case of RUN U, the signal RNU-P is asserted, and goes via the backplane to the Address Latch and Bank Select Module where memory address 8XXX is forced and starts the Utility program.

When the SINGLE STEP switch is set, depression of RUN U or RUN P results in the RUN mode for one machine cycle, stopping between TPA and TPB. Successive depressions of a RUN switch will cause execution of the program, one machine cycle at a time.

Inputs for external manipulation of the control modes are provided. EXT WAIT-P and EXT CLEAR-P are available on the backplane connector. Each is provided with a pull-down resistor and through one OR gate directly controls WAIT and CLEAR, e.g., a high on EXT WAIT-P causes a low on WAIT-N.

An Idle detector circuit counts S1 states to determine when 3 or more sequential S1 states occur. When Idle is detected, the RUN-N line will go high extinguishing the RUN light on the panel. The operator can terminate the Idle state by depressing RUNP. Then a single DMAOUT request will be made, and following the S2 state, processing will resume with the instruction following the Idle. The RUN light will be turned on when the program starts running again.

The Data Bus and Address Bus as well as WAIT-N, CLEAR-N, SC0-N, SC1-N and Q-P are sent to the display panel via J1. Each of these lines is buffered.

An interface connector J2 is provided for attachment of the optional Microterminal. Control switches from the Microterminal are electronically paralleled with those from the Control Panel.

Address Latch and Bank Select Module CDP18S206

The Address Latch and Bank Select Module stores and decodes the high-order byte of the memory address for use by all memory modules. Fig. 9 is a block diagram of this module. A(7:0) from the CPU module is latched into an eight-bit register by TPA. The outputs of this register, A(15:8), are provided on the backplane connector and to a one-of-16 decoder. The decoder outputs are 16 Bank Select lines, BS(F:0), which go to the backplane connector.

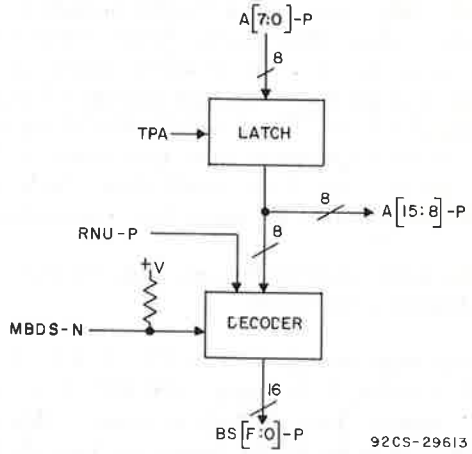


Fig. 9 — Address Latch and Bank Select Module CDP18S206 block diagram.

These signals break up the 65-kilobyte memory field into sixteen blocks of 4-kilobytes each. The supplied 4-kilobyte RAM module is wired to the lowest-order bank-select signal, BS0-P, so that it is located starting at address 0000. The ROM/RAM module is wired to BS7-P locating its starting address at 8000. RNU, a signal derived from the RUN U control panel switch, causes the decoder to see A15 as true. Thus, after a RESET, depression of RUN U will cause the starting address to be 8000, which is the location of Utility software.

Memory Bank De-Select (MBDS-N) is an input provided so that all Bank Select lines can be inhibited. A pull-up resistor is provided so that if no connection is made to this pin, the line is false, and the Bank Select decoder operates normally. This input may be used to logically disconnect the memory system when another is to be substituted.

Memory address lines A(7:0)-P are printed on the backplane from the CPU module to the Address Latch and Bank Select module and to all memory locations. A(12:8) address lines are printed on the backplane, from the Address Latch and Bank Select module to all memory locations. A(15:13) are available at the backplane connector and may be wired by the user if needed. The Bank Select lines, BS(F:0), are available on the backplane connector and must be wired to appropriate memory modules to suit the user's system organization. Refer to Section "Memory Addressing and Expansion" for information on adding additional memory to the CDS.

I/O Decode Module CDP18S509

The purpose of the I/O Decode Module is to provide decoded N-bit addresses for all I/O devices and to provide a two-level I/O addressing capability. Fig. 10 shows the functional block diagram of this module. The three N lines are decoded into 7 lines, N = 1 through N = 7. Because N = 0 is not a valid I/O operation, the zero decode is not used.

An output port latches a byte from the Data Bus when instruction 61 is executed. The eight latched bits are provided to the backplane connector as select lines SEL0-P through SEL7-P. The contents of the output port may be read, using input instruction 69, through the input port.

The select lines are used as follows. Each device (or group of devices) is assigned a unique eight-bit code. When the device sees its code on the select lines, it responds to I/O instructions. Thus, each code in the select lines allows a device (or group) to use six input and six output instructions. The 61 and 69 codes are excluded because they are dedicated to the group-select function.

In most systems, it will be sufficient to restrict the select code, by software convention, to a one-out-of-eight code. Then, each device need look at only one of the select lines. This technique allows for 48 unique output and 48 unique input instructions.

Automatic means are provided for enabling and disabling two-level I/O. The control flip-flop, when set, enables the chip select input of the I/O ports. The flip-flop is set by depressing the RUNU button and reset by depressing the RESET button, or under software control (by UT20). Note that the RESET button will also clear the Selection Register.

The control flip-flop can be locked out by wiring pin 9 of slot 13 to ground. With pin 9 grounded two-level I/O is always enabled and reset. RUN P will start a two-level I/O program correctly.

The select lines SEL(7:0)-P are available on the backplane. SEL0-P is prewired to the terminal (slot 14) and disk (slot 24) interfaces. Others can be wired to additional user I/O controllers as desired. For a further discussion of this subject, refer to "Two-Level I/O" under **Input/Output Interfacing** in the next Section.

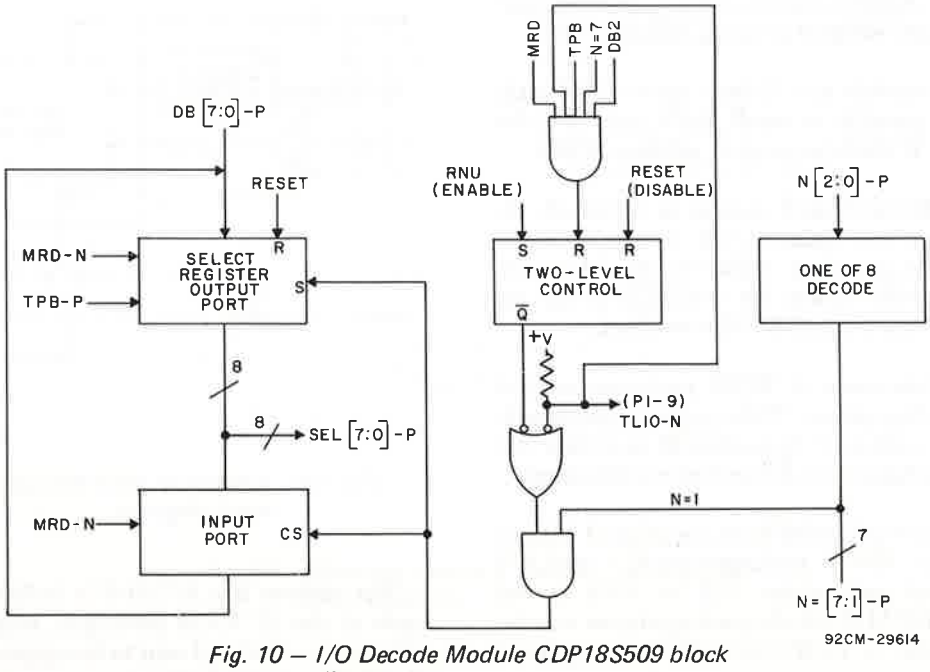


Fig. 10 — I/O Decode Module CDP18S509 block diagram.

ROM/RAM Module CDP18S401

The primary function of the ROM/RAM Module is to hold the system utility software. See Fig. 11. For this purpose, two socketted 24-pin positions are provided. Each of these locations accepts either 512-byte or 1024-byte ROM packages. Also, these ROM packages may be light-erasable EPROM's or mask-programmed CDP1832 or CDP1834 ROM's.

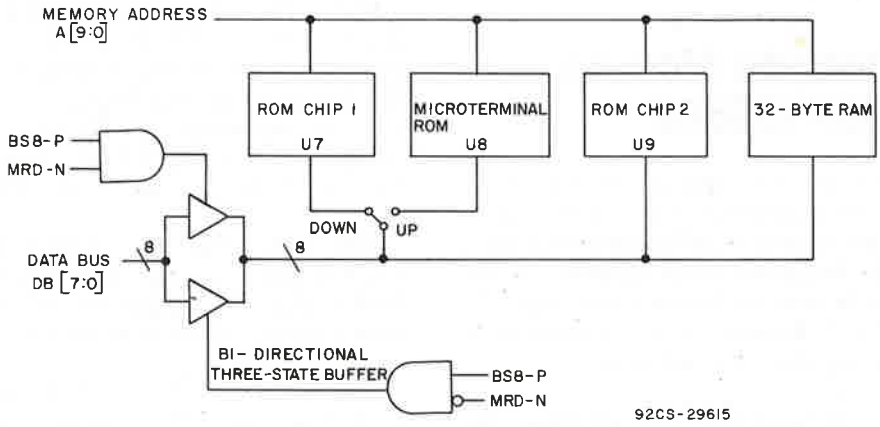


Fig. 11 – ROM/RAM Module CDP18S401 block diagram.

Pin X of this module location is wired to signal BS8-P from the Address Latch or Bank Select so that the utility program starts at memory address 8000.

Also on this module is a 32-byte static RAM chip (CDP1824) to provide a small work area for the utility software. It starts at memory address 8C00.

When the Microterminal option is installed, its ROM is inserted into location U8. A toggle switch selects either the standard utility or the Microterminal utility ROM. When the switch is in the up position, the Microterminal ROM is enabled.

Various combinations of ROM packages require specific link configurations. Refer to the detailed logic circuit and the tables of Appendix D to define the required link configuration for a given combination.

The PC board is provided with pre-printed links to select 1024-byte ROM packages in U7 and U9 (standard utility) and either 512 or 1024 in the Microterminal ROM (U8); the package types may be a 2708, CDP1832, or CDP1834 with the pre-printed links.

4-Kilobyte RAM Module CDP18S205

The 4-Kilobyte RAM Module, diagrammed in Fig. 12, provides the basic static RAM storage in the CDS. One module is supplied with the system, and memory may be expanded by adding modules in the memory expansion area provided. These modules are identical, but are assigned to the appropriate addresses by a Bank Select line wired to pin X of the added module(s). The basic RAM devices are 256 x 4 NMOS chips.

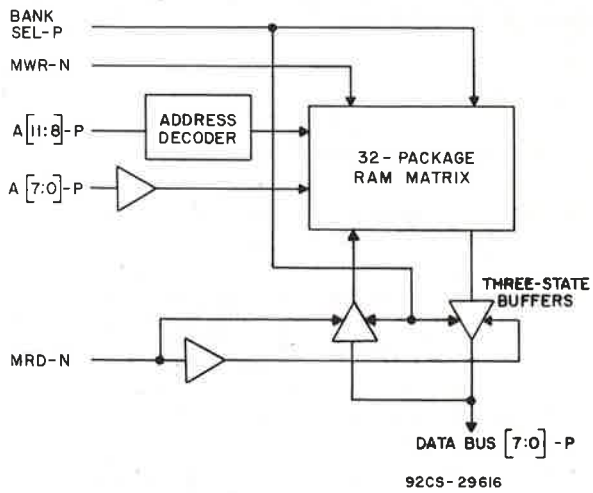


Fig. 12 – 4-Kilobyte RAM Module CDP18S205 block diagram.

The address bus A(7:0)-P is buffered and wired to each of the 32 RAM packages. High-order address bits are decoded and sent to the appropriate groups of RAM chip-enabled inputs.

The data-in and data-out lines from the RAM packages are buffered onto the system Data Bus, DB(7:0), through tri-state gated buffers. The direction is controlled by the Memory Read signal, MRD-N, and all buffers are enabled when Bank Select is true. Thus, when the module is not selected, it presents a high impedance to the Data Bus, minimizing loading effects.

Terminal Interface Module CDP18S507

The Terminal Interface Module, in conjunction with serial/parallel conversion routines in the Utility Program, provide an ASCII interface to a bit-serial terminal. The module, diagrammed in Fig. 13, has two interfaces, logically identical but electrically different, for two types of serial interface electrical conventions. The J1 connector provides a 20-mA current loop interface and a paper-tape reader control for a Teletype or similar terminal. The J2 connector provides an EIA interface for the J1 terminal, or others designed to the EIA RS232C specifications. Two cables supplied with the systems provide mechanical compatibility to either a TTY or an EIA interface.

The serial ASCII characters output to the terminal are formed by strobing Data Bus bit 0 into a D-type flip-flop under control of the UT20 TYPE routines. A 67 output instruction is used to generate the strobe. Bit 7 of the Data Bus is used to set the paper-tape reader control flip-flop and bit 6 to reset it via the same strobe.

Data from the terminal is transmitted to the Utility program via External Flag 4. Both the strobe and EF4-N are conditioned by a select signal, SEL-P. This signal may be wired to the I/O Decode module-select outputs, or, if left open, a pull-up resistor makes the select true all the time.

Pressing RESET on the control panel sets the data flip-flop and resets the paper tape control flip-flop. The output quiescent levels are then as follows:

EIA - voltage low (approximately -5V)
PT RDR - voltage low (approximately 0V)

Contact closure on the 20-mA loop incoming data line produces a low (true) on EF4-N. A low on the EIA incoming data line produces a low (true) on EF4-N.

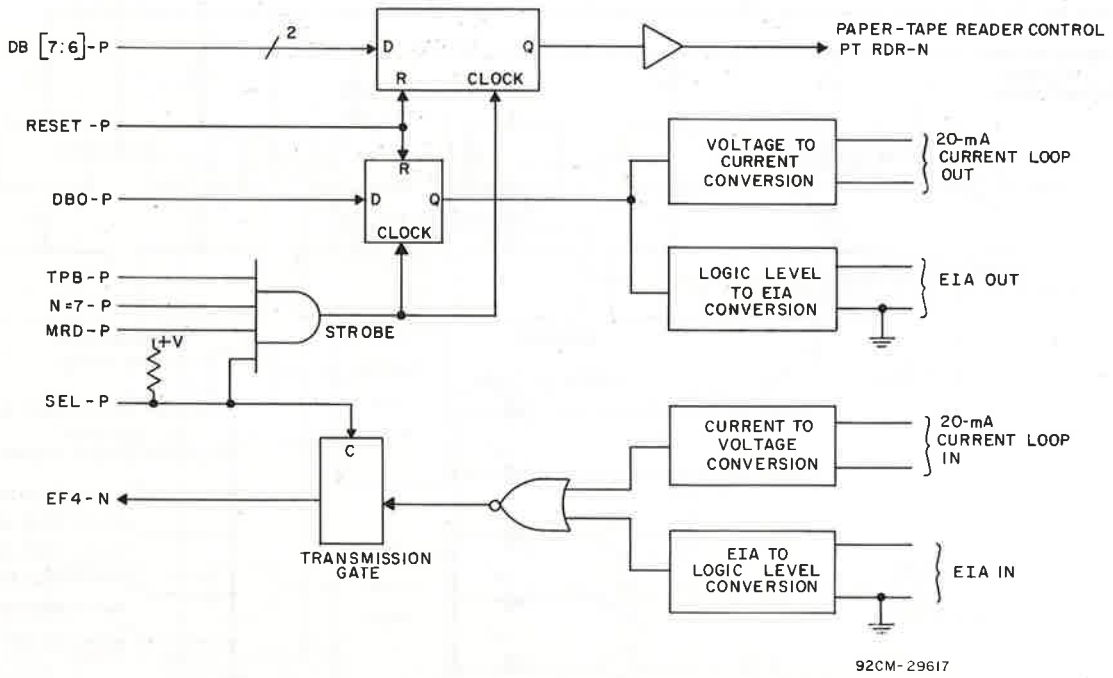


Fig. 13 — Terminal Interface Module CDP18S507 block diagram.

Display Board

The display board mounts on the hinged control-panel assembly. It contains six hexadecimal digit display units, six single LED displays, and six switches whose threaded bushings provide the method for mounting the board to the panel.

Communication with the Control Module, by way of connector J1 and a flat cable, provides all logic signals required. Power and ground are supplied to this board by wires soldered into plated holes provided. This power is always +5 volts even if the rest of the system is supplied from another voltage.

Fig. 14 depicts the logic on the board. Each display digit is driven by a LATCH-DECODER chip which take 4 bits from the address, or data bus, latches them if required, and re-encodes them into seven segments for the display. The front panel hexadecimal display presents the characters 0 through F.

The leftmost two digits latch the high-order memory address at TPA. The next two present the address bus continuously to provide the low-order address byte.

It must be remembered that the address is always displayed whether or not memory is being accessed. In the S0 state, the address is the location of data being fetched. In the S1 state, the memory address may or may not be significant, depending on the instruction being executed.

The rightmost two display digits show the Data Bus contents continuously when the control switch is in the BUS position. When the switch is in the LAST I/O position, the control module is conditioned to provide a latch pulse at TPB of any I/O operation.

The six single LED's display the state of the WAIT and CLEAR lines to the CPU, the state codes SC0 and SC1 from the CPU, the Q output from the CPU, and the system RUN state. These six displays are lighted when the specified condition is logically true.

Momentary action switches provide RESET, RUN U, RUN P, and LOAD functions. Toggle switches select SINGLE STEP or CONTINUOUS mode and LAST I/O or BUS display. The contacts of these six switches are sent directly to the Control Module.

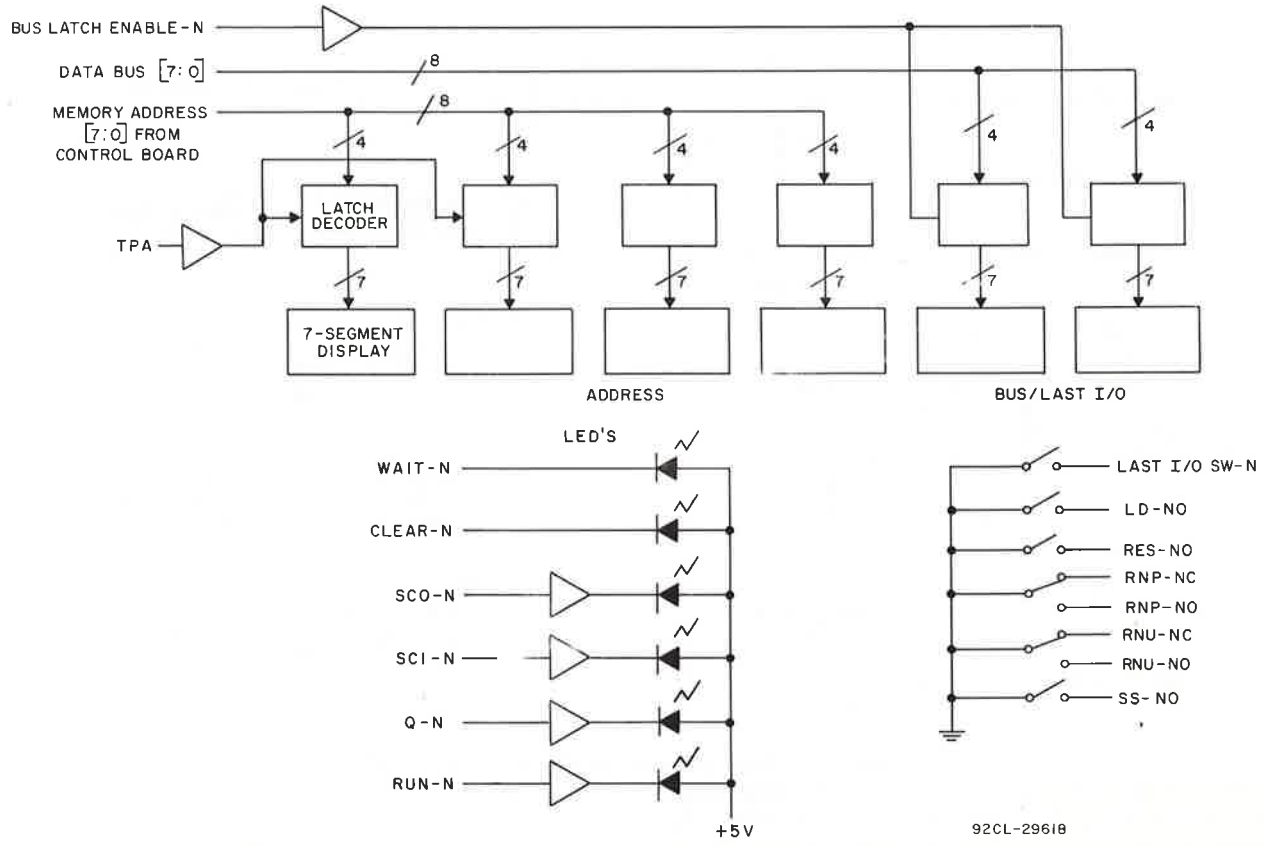


Fig. 14 — Display board block diagram.

Disk Interface Module Option CDP18S813

Slot 24 of the CDS is reserved for the optional Disk Interface Module. This module communicates with the disk drive unit through three byte I/O ports, buffers, and a 50-conductor flat cable. Its block diagram is given in Fig. 15.

The first output port, used for commands to the drive unit, is written to by a 65 output instruction with the I/O select register set to 01. Seven command bits are sent to the drive unit through a grounded-emitter open-collector transistor driver. The least significant bit is gated by the service request (SR) generated by the output port to provide a clock strobe.

Data is sent to the drive unit through a second output port loaded by a 64 output instruction. All eight bits are transmitted through grounded-emitter, open-collector transistors.

Data or status information from the drive unit is received by an input port, which is read by a 6E instruction.

Microterminal Option CDP18S021

Provision has been made to allow installation of the Microterminal and its ROM into the CDS. The Microterminal consists of a hand-held keyboard and display unit, its cable and mating connector, and a ROM containing a utility program UT5. A photograph is given in Fig. 16.

To install the Microterminal:

1. Turn power off.

2. Extract the ROM/RAM module from slot 9.

3. Install the ROM in the empty 24-pin socket U8. (The center of three 24-pin sockets).

4. Re-insert the ROM/RAM module into slot 9.

5. Insert the Microterminal cable connector into J2, the 20-pin connector on the outboard side of the Control module in slot 25. Carefully observe proper polarity by matching the index arrows on the connectors.

6. Set the switch on the ROM/RAM module to its up position.

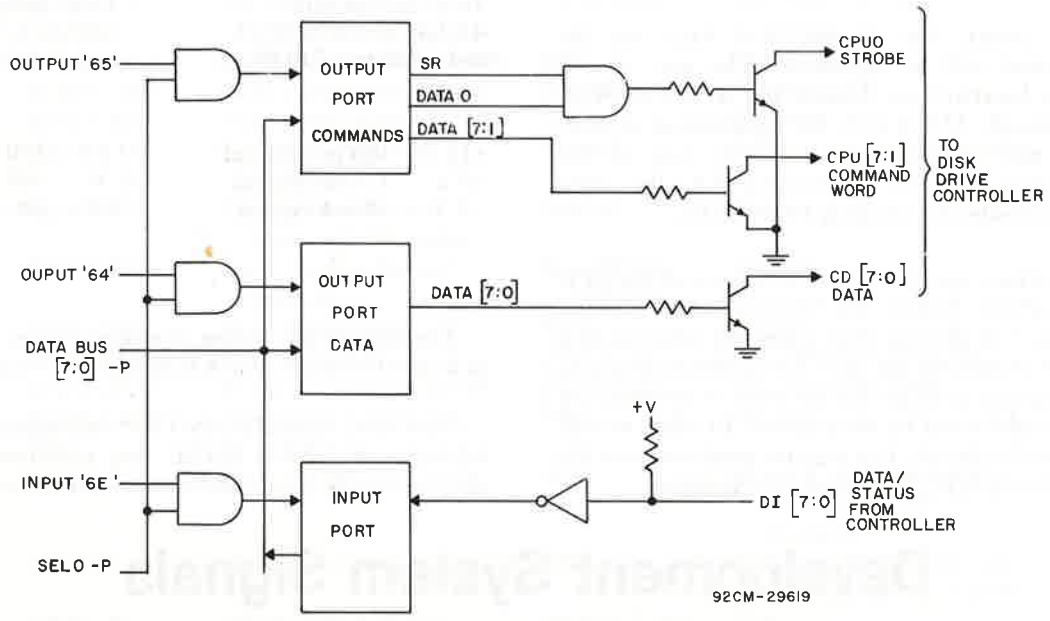


Fig. 15 — Disk Interface Module CDP18S813 block diagram.

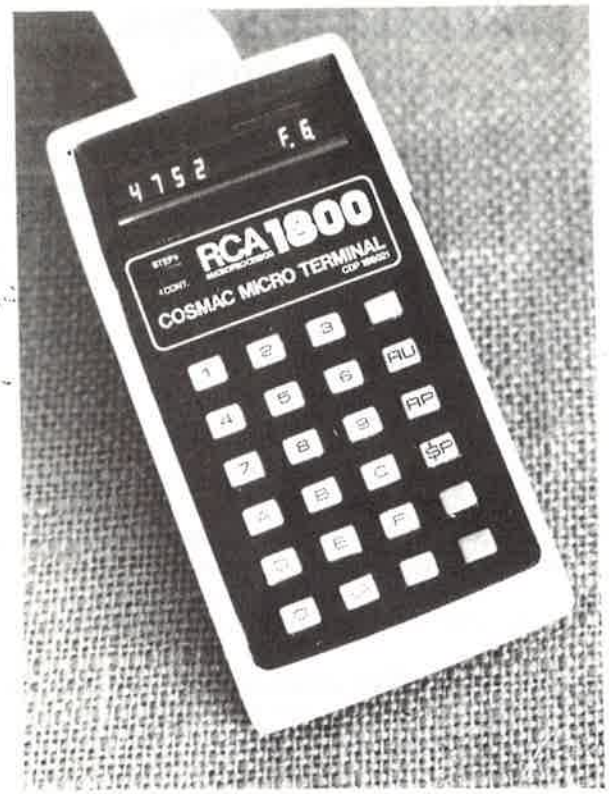


Fig. 16 – Photograph of RCA COSMAC Microterminal CDP18S021.

7. Turn power on.

8. Press RESET, then RUN U on either the Microterminal or the CDS front panel.

At this point, the display and keys on the Microterminal will be operative. The user should refer to the **Instruction Manual for RCA COSMAC Microterminal**, MPM-212, for operational details. Although that manual refers to the use of the Microterminal with the Evaluation Kit, the same operation considerations apply to the CDS.

A helpful note may be added to the use of the CPU register readout. Follow the Microterminal Manual instructions, but observe that when the contents of a register are displayed on the CDS address displays, the Bus displays provide the contents of the memory location as addressed by the register. In other words, M(R(N)) is displayed. For register read-out, use the front panel switch RUN U for single stepping.

To return to standard utility operation, set the switch on the ROM/RAM module to its down-position. This switching may be done while power is on, but if the Microterminal is to be disconnected, power should be turned off first.

Power Supply Module

The Power Supply Module consists of a transformer, rectifers, voltage regulators, pass stages, and heat sinks. The circuitry and heat sinks are mounted on a printed circuit board, which is mounted on the transformer bracket. This module is installed in the CDS chassis in the space beside the plug-in logic modules, occupying slot locations 26 through 32. See Appendix D, Fig. D19 for its circuit diagram.

AC input taps are provided for nominal voltages of 100, 115, 220, 230, and 240 volts rms, 50/60 Hz. The input is fused at 1.25 amperes when operated at 100 or 115 volts. It is recommended that the fuse be changed to 0.75 amperes when the CDS is operated at 220, 230, or 240 volts. Three output voltages provided are: +12 V at 0.5 A, +5 V at 6 A, and -5 V at 0.5 A.

The major current drain in the CDS is due to memory boards. The memory board current requirements are given below.

Total System with 4-kilobyte NMOS RAM and 3-kilobyte PROM:		Total System with 4-kilobyte SOS RAM and 3-kilobyte PROM:	
+12 V	100 mA typical	+12 V	100 mA typical
+5 V	1.5 mA typical	+5 V	800 mA typical
–5 V	60 mA typical	–5 V	60 mA typical

The remaining system components are all CMOS and take less than 50 mA from the +5 V supply.

Note that, typically, the CDS can support up to 12 kilobytes of NMOS RAM. Any additional memory may have to be powered from an external supply.

Development System Signals

Table IV summarizes all of the CDS signals. For each signal, the source and destination modules are listed. The term 'USER' in a column designates a signal that may be derived from or sent to user-added devices.

TABLE IV – CDS SYSTEM SIGNALS

Signal Name	Description	Source	Destination
A[7:0]–P	Low-order memory address bits	CPU	Memory, Address Latch Control
A[15:8]–P	High-order memory address bits	Address Latch	Memory
ANY I/O–P	OR'ed output from N-lines	CPU	Control
BS[F:0]–P	Memory bank select	Address Latch	Memory
CDO[7:0]–N	Data bits from disk interface module	Disk interface module	Disk Drive
CLEAR–N	CPU clear signal	Control	CPU, Disk
CLEAR SW	From Microterminal	Microterminal	Control
CPU[7:0]–N	Command bits from disk interface module	Disk interface module	Disk Drive
CPU PWR	$+V_{DD}$ supply to CPU	Power Supply/ User	CPU
DB[7:0]–P	Bidirectional data bus	CPU	Memory, I/O, Control, Terminal
DI[7:0]–N	Data bits from disk drive to disk interface module	Disk Drive	Disk interface module
DMAI–N	DMA IN request	User	CPU
DMAO–N	DMA OUT request	User, Control	CPU
EF[4:1]–N	Flag inputs to CPU	I/O, Terminal, Control	CPU
EX CLK	External clock input	User	CPU
EX WAIT	External wait to CPU	CPU	User
INT–N	Interrupt request	User	CPU
LOAD SW	From panel switch	Panel	Control
MBDS–N	Memory bank deselect	User	Address Latch
MRD–N	Memory read signal	CPU	Memory, I/O, Control, I/O Decode, Terminal
MWR–N	Memory write signal	CPU	Memory
N[2:0]–P	N-lines from CPU	CPU	I/O Decode, Control
N=[7:1]–P	Decoded N-lines	I/O Decode	I/O, Terminal, Disk, Control
Q–P	Single bit output from CPU	CPU	Terminal, I/O
RESET–OP	Reset signal from control	Control	Memory, I/O
RESET SW	From panel switch	Panel	Control
RNU–P	Signal to run utility program	Control	Address Latch, I/O Decode
RUN–N	Signal indicating continuous S0,S1 cycles	Control	Display
RUN P SW	From panel or Microterminal	Panel/Microterminal	Control
RUN U SW	From panel switch	Panel	Control
SC0[1:0]–P	State code lines	CPU	I/O, Terminal, Control
SEL[7:0]–P	Two-level enabling signals	I/O Decode	I/O, Terminal, Disk
SINGLE-STEP SW	Single-step control input	Panel	Control
TPA–P	Early pulse in CPU cycle	CPU	Memory, Address Latch, I/O Decode, I/O, Control Terminal
TPB–P	Late pulse in CPU cycle	CPU	Memory, I/O Decode, I/O Control, Terminal, Disk
UA15–N	OR'ed output of A15 and RNU	Address Latch	User
WAIT–N	CPU wait signal	Control	CPU

Memory Addressing and Expansion

To aid the user in interfacing the COSMAC Development System to added memory hardware, this section discusses memory module addressing and the use of custom memory modules.

Memory Organization

The total directly addressable memory space (65,536 bytes) may be considered as being divided into 16 banks of 4096 bytes each. The 16-bit memory address, A(15:0), can be divided into two fields: A(15:12) being a bank number and A(11:0) selecting the byte within a bank, (see Fig. 17). A bank may be further subdivided into four blocks of 1024 bytes each (in which case A(11:10) is a block number within a bank and A(9:0) is the address of a byte within the selected block) or into eight blocks of 512 bytes each (A(11:9) defining the block number and A(8:0) identifying the byte within the selected block).

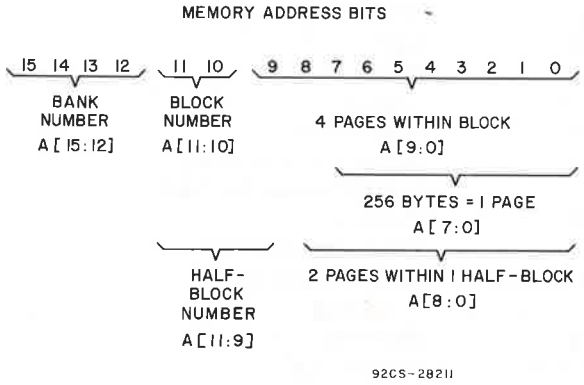


Fig. 17 – Memory address bit assignments.

Each memory plug-in module comprises a bank or a block of memory. When a module is inserted in a given plug-in slot, external wiring of appropriate pins on the connector in that position defines its bank and block numbers (the range of addresses to which it responds). Thus, users may define arbitrary address ranges for the memory modules they use.

The COSMAC Development System CDP18S005 is supplied with a 4-kilobyte RAM module wired to occupy the lowest memory address range (starting at address 0000). The CDS is also equipped with a ROM containing the Utility Program UT20. UT20's address range is 8000 to 83FF. Another ROM containing the disk loader program occupies addresses 8400 to 87FF. UT20 also uses a dedicated RAM of 32 bytes starting at address 8C00. Refer to Table III for a list of module position assignments.

RCA Modules

Each RCA memory module includes a sufficient number of enable inputs which can be used to assure that it will respond only over its assigned address range. Every module has a bank select input at pin X. When this signal is high, the module is ACTIVATED or ENABLED. When it is low, the module is DISABLED or DESELECTED. Whenever a module is added or moved in the address space, this overriding SELECT input must be connected to the proper enabling source.

In the Development System, the supplied ROM memory is enabled by BS8-P, so that the UT20 program begins at address 8000. The RAM space, on the other hand, is designed to begin at location zero. See Fig. 18 for a CDS memory map. The memory bank select module decodes the upper four address bits to provide sixteen RAM bank-select signals. The supplied 4-kilobyte RAM (in slot 8) is enabled at pin X by the lowest of these outputs, called BS0-P.

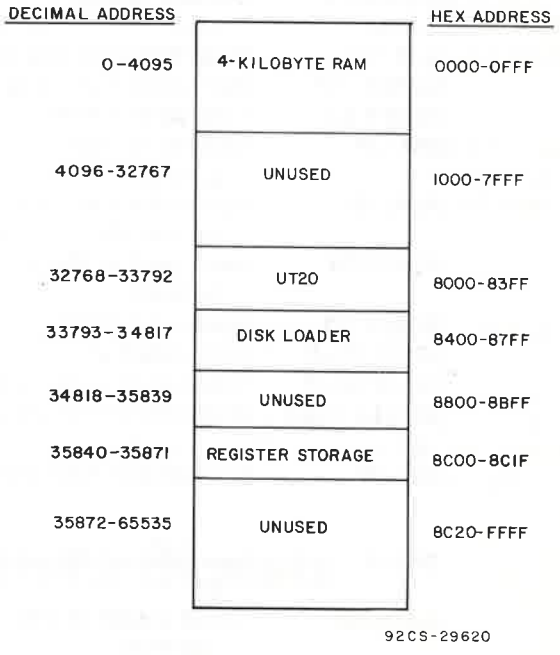


Fig. 18 – CDS CDP18S005 memory map.

For this Manual a **page** is defined as 256 bytes, a **bank** as 4096 bytes, and a **block** as either 512, 1024, or 2048 bytes, as convenient. The memory bank select module supplies signals which can be used to locate a memory module in any available bank. The appropriate memory bank select signal (BS(F:0)-P) should be connected to pin X of any inserted memory module. If the inserted module is a 4-kilobyte memory, no additional selection wiring has to be done.

NOTE: **Added RAM below address 8000 must be contiguous for the Editor and Assembler to operate correctly.**

Reviewing the memory addressing scheme, buffered address bus lines from the CPU, A(7:0) are latched in the Address Latch and Bank Select module, and also wired to all memory slots. Once latched at TPA, the latch outputs become the high-order address byte, A(15:8). Of these, A(12:8) are pre-wired to all memory slots; A(15:13) are available for custom wiring, if required. A(15:12) are decoded into 16 Bank Select lines, BS(F:0), which are available for wiring to any module.

When memory or I/O is expanded, care must be taken to budget power consumption to stay within the system's power availability. Otherwise, additional power supplies may be required. Further information may be found under "Power Supply Module" earlier in this Section.

Custom Memory Modules

RCA will continue to offer new RAM and ROM memory modules designed specifically for use with the COSMAC Microprocessor. Users who try modules of their own design and construction with the COSMAC Development System should, of course, observe its physical and electronic constraints. The physical constraints are fairly obvious. If the memory is to reside in the CDS, the cards containing it should be no more than 6.5 inches deep and no more than 0.4 inch in thickness over-all if adjacent slots need to be occupied. If the memory is to be external and serviced by a cable, then the cable connector may plug into a Memory Bus slot. This arrangement may require the user to buffer the address bus (A15-P to A0-P) and the data buses (DB7-P to DB0-P), depending on the length of cabling and the drive required. The memory must also have bidirectional I/O capability so that it can be hung on the data bus. If it does not, appropriate three-state buffers must be incorporated on the module.

The COSMAC CPU architecture does not require memory cycles to be contiguous, i.e., to immediately follow each other in time. (For further information on memory timing refer to the material on "Memory Interface and Timing" in the section "Interfacing and System Operations" in the **User Manual for the CDP1802 COSMAC Microprocessor, MPM-201.**) Thus, the cycle times of added memory modules are generally not critical. They must, of course, be less than the CDS cycle time of approximately four microseconds. What are important, however, are the access and write times. The read access time in the CDS environment, at a clock frequency of 2.0 MHz, should be 0.6 microsecond maximum, and the memory should take no longer than that time to write. Slower memories may be incorporated by supplying a slower clock, or using a "hand-shaking" technique via the WAIT input of the CPU.

Input/Output Interfacing

One of the fundamental advantages of the COSMAC architecture is the richness of the CPU-I/O interface. A significant number of interfacing "resources" (many I/O-oriented signals, with many different functions) are available for use. These resources include the DMA and INTERRUPT request lines, the four EXTERNAL FLAG input lines, the three N output lines, the related control signals (CPU STATE and TP's), and, of course, the I/O data bus. A wide variety of interfacing techniques are possible, limited only by the imagination of the designer. Only a few are discussed below, each supported by an illustrative example.

Module Enable Philosophy

A fundamental interfacing feature of each of the RCA-supplied modules is the inclusion of at least one over-all "enable" input signal. Generally, this signal is "pulled" to the enabled state (high) by a resistor on the board, so that if its connector pin is left alone, the module will be permanently enabled. This input can also be driven by a signal, possibly derived from a manual switch, so that the module can be selectively enabled. This feature was discussed in the previous section "**Memory Module Addressing**".

The broad function of disabling an I/O-oriented module is to temporarily or permanently decouple it in some way from the I/O system. As will be discussed by specific examples, each I/O-oriented CPU input pin can be considered as the common destination of a set of wire-OR'ed (bussed) signal sources - only one of which is asserted at a time. In this case, the various I/O enable signals are used to assure that only one source at a time is coupled to the CPU input pin. For example, the terminal interface module includes an enable signal so that when it is disabled, EF4 is available for use by any other devices in the system.

The use of such enables in the case of programmed-I/O data transfers is discussed in detail in the **User Manual for the RCA CDP1802 COSMAC Microprocessor**, MPM-201. During the execution of an I/O instruction, the information present on the three N lines of the CDS and on the eight data bus lines can be interpreted in any one of several ways. In a one-level I/O system, the N code selects one of a set of devices, while the data bus carries information between that device and memory. In this case, up to seven input devices and up to seven output devices are possible when used in conjunction with the MRD signal to specify the data direction. The I/O decoder module is used to generate an individual I/O command "strobe" to activate the selected device. The command strobe from the I/O decoder causes it to put data on the I/O bus or to latch data from it.

It should be noted that a "sub-one level" I/O system could be used, for a sufficiently simple system, in which, for example, one bit of the N code (appropriately AND'ed with a TP) can act directly as a command strobe. For that matter, if only one I/O device not requiring data transfer is used, the Q output can be used to activate it.

For more complex systems, a two-level I/O approach is used. This approach consists of two steps; first enabling and then activating the selected I/O device. The CDS has two-level I/O capability built in as a user convenience. It is described in the next section.

Two-Level I/O

The I/O Decode Module in slot 13 provides seven lines, N=1 through N=7 which may be wired to the I/O slots to define a unique N-value address. Combined with the MRD signal, these signals define all I/O instruction codes. For example, $N=1 \cdot \overline{MRD}$ defines the 69 input instruction, while $N=1 \cdot MRD$ defines the 61 output instruction.

In addition, the I/O Decode module contains an output port that latches the Data Bus on a 61 instruction. The latched outputs, SEL(7:0), are available for custom wiring to I/O slots. This feature is used for two level I/O addressing where more than seven Input/Output instructions are required.

Suppose a module is wired to SEL2 and N=3. Then a 61 instruction whose data transmitted = (0000 0100) selects that module. A subsequent 63 or 6B instruction would be interpreted as an output or input instruction intended for that module. All other modules are de-selected and ignore the I/O instructions. The Select lines remain set until changed by another 61 or the RESET switch is activated.

Thus, not counting the 61 and 69 instructions, a total of six 'N' lines times 8 Select lines provides 48 unique input and 48 unique output decodes for I/O devices. Should more codes be required, the 8-bit Select register may be decoded to provide up to 255 Select numbers. In this case, a decoder must be provided by the user.

Of course, software conventions must be consistent with the system hardware architecture. Also, when using the two-level I/O addressing scheme, all I/O modules must be designed and wired to work with the selection convention chosen.

Care must be taken when expanding the system I/O capability to stay within available power limits. Further information may be found under "Power Supply Module" earlier in this Section.

The Terminal Interface module and optional Floppy Disk Interface module are already wired to SEL0-P. This selection signal is automatically controlled by UT20 and should not, in general, be used by user-added I/O devices. Refer to Fig. 10 for the following discussion.

The CDS is delivered with a jumper wire grounding TLIO-N (pin 9, slot 13). This jumper enables two-level I/O selection at all times. A 61 instruction is used to latch data into the selection register. UT20 automatically writes a 01 to the selection register whenever it is started, enabling the terminal and floppy disk interfaces. A RESET will

force all zeroes into the register. Instruction 69 is used to read back the contents of the register for use by interrupt-handling subroutines.

Users who are developing systems having one-level I/O should remove the jumper on TLIO. The system will then work as follows. A 61 instruction is used to enter new selection data so long as the two-level control flip-flop is set. This flip-flop is set when the RUN U key is pressed. UT20 immediately writes a 01 to the selection register, enabling the terminal interface. The flip-flop is reset by the RESET switch or by execution of the $P command, which additionally writes a 00 to the selection register. With the control flip-flop reset, the 61 and 69 instructions are free for use like any other instruction and cannot be used to control the selection register.

Several operational considerations exist for the two options of TLIO grounded or not. These options are summarized below.

With TLIO-N Grounded:

A. **RESET, RUN P Sequence**

1. Starts execution at location 0000.
2. Two-level I/O permanently enabled.
3. Instructions 61 and 69 reserved for two-level selection.
4. Program starts with P=X=0, IE=1, Selection Register=00.

B. **RESET, RUN U Sequence**

UT20 starts and runs with selection group 01 enabled.

$U Command

1. Program starts at location specified.
2. Two-level I/O permanently enabled.
3. Instructions 61 and 69 reserved for two-level selection.
4. Program starts with P=X=0, IE=1, Selection Register=01.
5. If an interrupt is pending at start-up, UT20 will report it.

$P Command

1. Program starts at specified location.
2. Two-level I/O permanently enabled.
3. Instructions 61 and 69 reserved for two-level selection.
4. Program starts with P=X0, IE=1, Selection Register=00.
5. If an interrupt is pending at start up, UT20 will attempt to report it.

With TLIO-N Open (High)

A. **RESET, RUN P Sequence**

1. Starts execution at location 0000.
2. Two-level I/O is permanently disabled.
3. All I/O instructions are available to user.
4. Program starts with P=X=0, IE=1, Selection Register=00.

B. **RESET, RUN U Sequence**

UT20 starts and runs with selection group 01 enabled.

$U Command

1. Program starts at location specified.
2. Two-level I/O enabled.
3. Instructions 61 and 69 are reserved for two-level selection.
4. Instruction 67 with data bit pattern (xxxxx1xx) must be avoided if two-level selection is to remain enabled.
5. Program starts with P=X=0, IE=1, Selection Register=01.
6. If an interrupt is pending at start-up, UT20 will report it.

$P Command

1. Program starts at location specified.
2. Two-level I/O permanently disabled.
3. All I/O instructions are available to user.
4. Program starts with P=X=0, IE=1, Selection Register=00.
5. If an interrupt is pending at start-up, UT20 will attempt to report it.
6. Starting a program in this case precludes the use of the devices controlled by the Terminal and Disk Interface modules.

NOTE: All RCA-supplied programs should be started with the $U command in all cases.

Interfacing Signals and Custom I/O Modules

User devices can be interfaced to the CDS with signals available at the I/O bus and from the I/O decoder. There are seven signals which are sensed by the CPU, namely, EF1-N, EF2-N, EF3-N, EF4-N, DMAIN-N, DMAOUT-N and INTERRUPT-N. These signals are pulled high with 22-kilohm resistors on the CPU board and are brought to the I/O bus. Control electronics for these signals should use a transmission gate which pulls the signal lines low

when activated and appears as an open circuit when not activated. Thus, several devices may be wire-or'd to these lines.

There are eight output data lines, DB0-P to DB7-P, which may be connected to user devices. Data here is valid at TPB of the I/O execution cycle. When an OUT N instruction (machine code 6N, N=1-7) is executed, the I/O decoder sends out decoded signals of N=1-P through N=7-P. These signals (plus any of the SEL0-P through SEL7-P, if desired) may be used to latch the data appearing on the data bus at the trailing edge of TPB.

All signals in the CDS swing between GND and +5 volts dc. Other MOS devices such as C-, N- or PMOS may be added by the user (if voltage levels are compatible) or bipolar devices such as TTL or low-power Schottky TTL devices. All data and address bus signals as well as the CPU signals are capable of driving one TTL load worst-case (sinking 0.2 mA at 0.4 volt). Open-collector devices should be used to drive the Data Bus, EF, DMA, or Interrupt lines, letting the 22-kilohm pull-up resistors on the CPU module generate the logic "one" voltage level.

However, a direct interface is not recommended. A better practice is to buffer all signals to and from a user-designed module through CD4049 (inverting) or CD4050 (non-inverting) or CDP1856/57 buffers. This technique is preferable to loading the various busses and possibly causing problems on another module. Also, maintaining a CMOS interface to the CDS encourages the good design practice of inserting 1-kilohm series resistors in all lines - at least initially. This practice will prevent accidental and costly destruction of components should the user-designed module not perform as expected.

For a list of all signal names and their meanings, refer to Table IV. Note that many I/O signals are preassigned and should not be used indiscriminately. Table V lists the instructions and flags used in the CDS and Table VI the reserved codes for future use.

TABLE V – PREASSIGNED I/O INSTRUCTIONS AND FLAGS

Group 1 [00000001] – Two-level I/O enabled.

Instruction		Action
61	OUT1	Latch MR(X) into Two-level Selection Register
63	OUT3	Output Segment Data to Microterminal
64	OUT4	Output Digit Data to Microterminal
64	OUT4	Output to Disk Interface (Data Bits)
65	OUT5	Output to Disk Interface (Control Bits)
66	OUT6	Line Printer Data Out
67	OUT7	Terminal Interface Serial Output Using DB0
67	OUT7	Terminal Interface Paper Tape Reader Control Using DB[7:6]
67	OUT7	Disable Two-level I/O when DB2 = 1
69	INP1	Read Two-level Selection Register
6E	INP6	Input Data Bits from Disk Interface
6C	INP4	Input from Microterminal
EF4		Serial Input from Terminal Interface
EF3		Keyboard Active Signal from Microterminal
EF1		Use for High-Speed Printer Interface Option

TABLE VI – RESERVED I/O INSTRUCTIONS AND FLAGS

Group 2 [00000010] – Two-level I/O enabled.

Instruction		Action
61	OUT1	Latch MR(X) into Two-level Selection Register
62	OUT2	Load UART Transmitter Holding Register
63	OUT3	Load UART Control Register
69	INP1	Read Selection Register
6A	INP2	Read UART Receiver Hold Register
6B	INP3	Read UART Status Register

There are typically 2 A at +5 V, 400 mA at -5 V, and 400 mA at +12 V reserve available from the CDS power supplies. This reserve should be adequate to handle most user-supplied additional boards. With series resistors in all interface lines, as recommended above, power sequencing should not be a concern.

Timing diagrams for DMA requests and INTERRUPTS are also shown in the**User Manual for the CDP1802 COSMAC Microprocessor, MPM-201.** As explained there, any DMA request or INTERRUPT will cause the CPU to make a transition out of the IDLE state (repetitive S1's), and cause program execution to start. The COSMAC CPU is sensitive to both DMA and INTERRUPT after RESET because it is in an IDLE state. Thus, CLEAR-N should be used to disable these external requests until they are required. User devices should not issue DMA or INTERRUPT requests until explicitly permitted by program or by logic, because initialization of CPU registers is necessary before INTERRUPTS and DMA's can be handled.

To complete this Section, the implementation of DMA, Interrupt, and basic serial and parallel I/O devices are discussed next. Reference should be made to the cited portions of the **User Manual for the CDP1802 COSMAC Microprocessor, MPM-201.**

DMA Input

Assuming only one I/O device needs the DMA input port, the arrangement shown in Fig. 19 is possible. Systems having more then one DMA channel (either 2 or more DMA-IN's or a mixed DMA-IN and DMA-OUT) must use the Interrupt facility to establish vectoring. Notice that if interrupts will not be used in the system of Fig. 19, the SC1-P can be used directly for the output port and signal flip-flop: it does not have to be gated with SC0-P. The MRD-N line is also not strictly necessary unless a mixed DMA-IN/DMA-OUT system is being constructed.

DMA Output

Assuming only one device is using the DMA-output channel, the circuit of Fig. 20 can be used. Again, note that multi-channel DMA systems must use the Interrupt facility. The discussion concerning SC1-P and MRD-N in the previous section also applies here. The Control Module uses DMA-OUT to terminate IDLE. Isolation of a user-supplies signal by means of a three-state device, diode, or the like is required on this line.

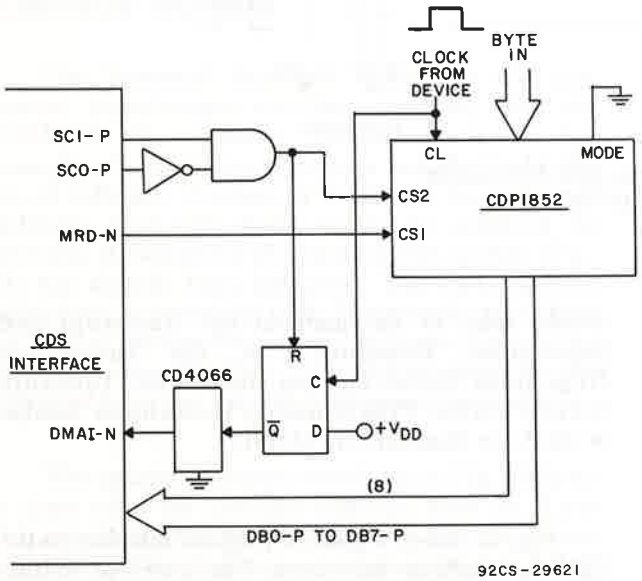


Fig. 19 – DMA input example.

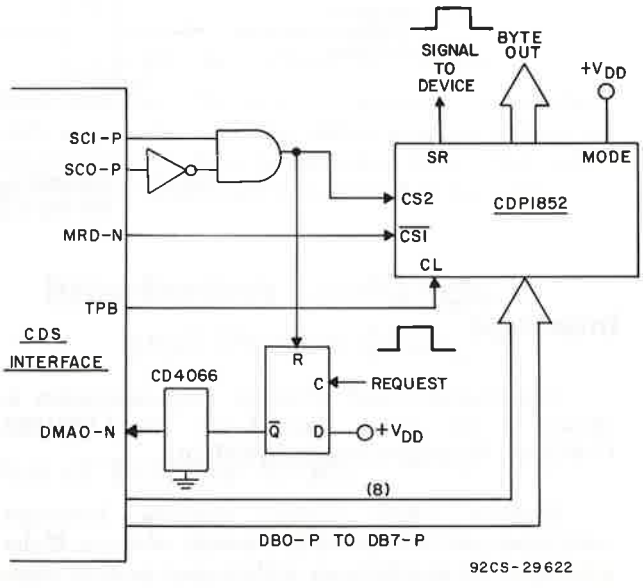


Fig. 20 – DMA output example.

Byte I/O

A general purpose Byte I/O Module designed for use in the CDS is shown in Fig. 21. By appropriate connection of the module's enabling signals to the 'N' lines and SELECT bits from the I/O Decoder, the module can be assigned to respond to any programmed I/O instruction. The module could be permanently enabled (by leaving the Select input open) or permanently disabled (by grounding the Select input), if desired.

The source of an asynchronous interrupt must be deduced from externally generated information, such as the contents of an interrupt status register (interrogated by an I/O READ instruction), any external flag values, or the address of register R0 as dictated by DMA activity. The CPU contains a programmable INTERRUPT ENABLE (IE) bit. Its operation is discussed in detail in the **User Manual for the CDP1802 COSMAC Microprocessor**, MPM-201. Before designing an interrupt-generating I/O circuit for installation in the CDS, the reader

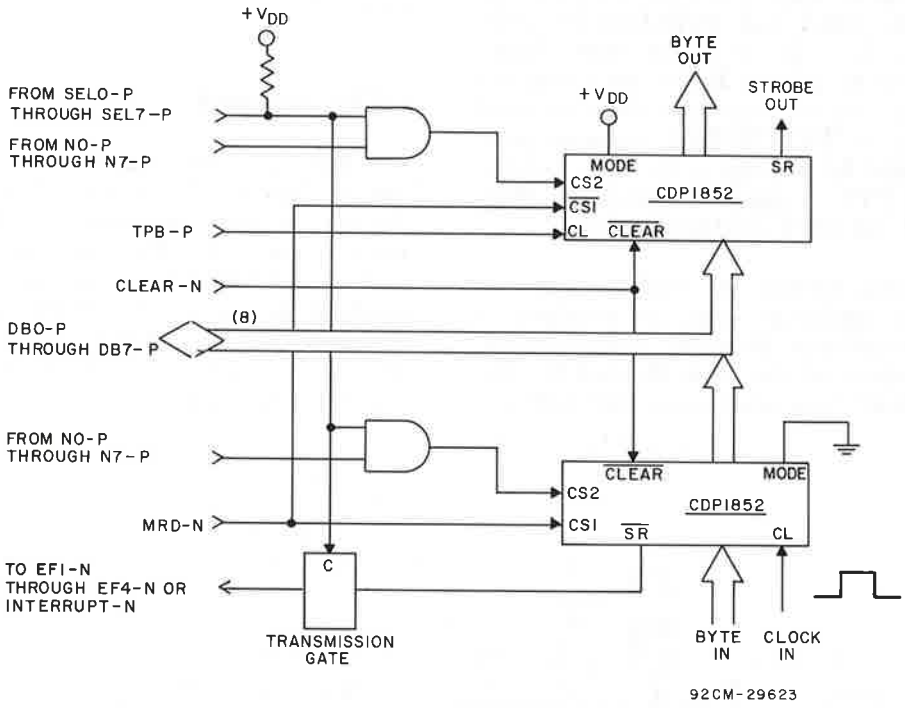


Fig. 21 — General-purpose byte I/O module for the CDS CDP18S005.

Interrupt

A straightforward interrupt implementation is shown in the **User Manual for the CDP1802 COSMAC Microprocessor**, MPM-201.

Systems which require multiple interrupt conditions can be handled in a variety of ways. If the interrupts are synchronous with respect to each other (i.e., there is a prior knowledge that there will be a specific patterns such as ABAB....or AABCAABCA....), then all the handling can be accomplished with software. The interrupt analysis pointer is merely re-initialized after each service to the address necessary to handle the next service.

should refer to the material on "**Interrupt and Subroutine Handling**" in the **Instruction Repertoire** Section and the material on "**Interrupt Service**" in the **Programming Techniques** Section of the **User Manual**, MPM-201.

Fig. 22 shows a general purpose interface to the CDS for multiple interrupts. The interrupt routine would issue a specific input instruction (say '62') and the device requesting the interrupt would put its address on the data bus where it can be examined by the CPU and used to vector to the right routine. Additional hardware would be required if the possibility of simultaneous interrupts exist.

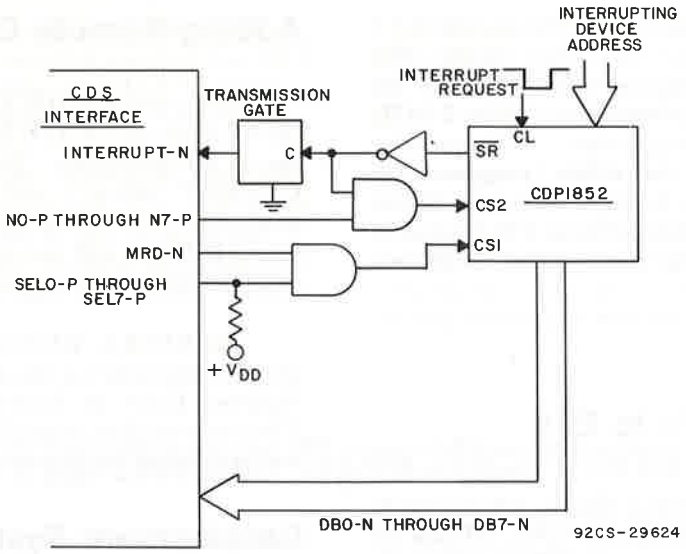


Fig. 22 — General-purpose circuit for vectored interrupts.

A good programming technique is to put the bytes 71 and 00 as the first two instructions in your program. That will disable interrupts until you are ready for them. At that time, they can be re-enabled by setting X=P, then performing the instruction sequence 70, (X,P) where the byte after 70 contains the initial values for X and P. Refer to the User Manual MPM-201 for interrupt servicing program techniques.

Bit Serial Interface - The Terminal Interface Module

The Terminal Interface Module is the only custom input/output interface supplied with the COSMAC Development System. It is another example of minimizing hardware complexity by the use of software. Further, it illustrates the increased flexibility that can more readily be achieved by software. A functional diagram for this module (Fig. 13) has already been discussed. The CPU receives serial data by sampling EF4. It transmits serial data via bit 0 of the data bus in conjunction with an output instruction, specifically 67. The detailed logic for the Terminal Interface Module is shown in Appendix D.

The sample character waveform in Fig. 23 helps to show what the interface software must do. Each character is framed by a START bit and one or two STOP bits. The character waveform signal is tied to EF4-N, sensed by UT20 at the midpoints of each of the bits, and assembled into the ASCII character. A character is transmitted one bit at a time with bit 0 of the data bus latched by a D flip-flop. The Q flip-flop can also be programmed to provide serial output data.

The flexibility obtainable with software is demonstrated by the ability of the program UT20 to sample a character string and adjust its timing so as to cope with terminals of different, even non-standard,

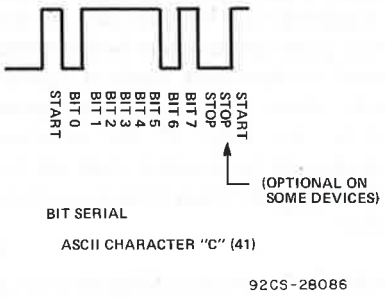


Fig. 23 — Sample character waveform.

character rates. However, it should be noted that while a program is timing either input or output in this manner (i.e., by counting instruction executions), the processor is completely dedicated to that task.

Interfacing Techniques and Precautions

Use of External Clock

Procedure: Remove the crystal on the CPU board. Connect pin P1-12, slot 12 (EX CLK) to the clock generator. The external clock signal should swing between +5 volts and GND with rise and fall time equal to or less that 15 microseconds. Because the COSMAC CPU is a static system, single-stepping (single clock cycles) or steps in bursts of 8 or 16 clock cycles, for example, are possible.

Clock frequencies higher than the standard 2.0 MHz may be employed in some cases. The COS/MOS interfaces supplied with the CDS (at V_{DD}=5 volts) will not operate above about 3 MHz because of the short timing pulses. At frequencies lower than 2.0 MHz, the utility program will eventually fail to time the terminal serial characters properly because of quantizing effects. The frequency at which these effects occur is a function of the terminal character rate.

External Flags EF1 to EF4

The external flags offer a simple, yet powerful, input interface to the COSMAC CPU. Means by which a program may test an external flag and branch conditionally on its value have already been discussed. The use of a flag as a bit-serial data input port was also described. Note that, with the terminal interface module in its slot, EF4 is unavailable for other devices unless the I/O data terminal-to-EF4 patch is disabled, by forcing SEL-P low. Because transmission gate outputs may be connected as "wire-ors", several devices may share a specific External Flag signal, when necessary. This arrangement is illustrated in the logic of the terminal interface module. It should be recalled that all four EF's are pulled up to V_{DD} through 22-kilohm resistors on the CPU module.

In order for an external flag to play a functional role in a COSMAC-based system, it must be tested by the program at the time when action is required. Further, in programs which incorporate a periodic flag test, there must be some means for the interface logic to sense that the flag stimulus has caused a response. In system where it is necessary to detect failure or error conditions, one or more flags may be used. If immediate action is required, the flags may be used as a means to vector interrupts.

Adding I/O Devices

When additional I/O is added to the CDS, it is most important to remember that certain I/O instructions and group selection codes are already assigned to specific CDS modules. These codes are detailed in Tables V and VI. Care must be taken not to overlap them. I/O can also be added in the memory field of the CPU for memory-mapped I/O functions. In that case, be sure to refer to the memory map, Fig. 18, to determine free memory locations.

Adding Remote Control

Along with the DMA, Interrupt, and Flag lines, the CLEAR and WAIT lines of the CPU are brought out to the backplane. These lines are labeled EXT CLEAR-P (P1-11) and EXT WAIT-P (P1-12), respectively, from the Control Module. The EXT CLEAR-P line will **not** reset the entire CDS system, but only perform the appropriate CPU functions.

The RESET, RUN U, and RUN P signal lines are also available on the J2 connector of the Control Module. Refer to Appendix D for pin numbers. These connections normally go to the Microterminal, but can also be used for other remote control inputs.

Development System Dynamic Characteristics

COSMAC CPU timing and dynamic specifications are to be found in the **User Manual for the CDP1802 COSMAC Microprocessor**, MPM-201, and in the data sheet for the CDP1802. COSMAC-based products should be designed to those specifications. The CDS interface is designed to faccilitate functional experiments. The interface supply voltage, V_{DD}, of +5 volts, the modular construction, and the consequent capacitive loading make it a slower system than that which can be supported by the COSMAC CPU itself.

The signals of shortest duration in the CDS are TPA-P and TPB-P. The TPB timing signals occur late in a machine cycle to indicate that the data present on the bus (from a memory access) is valid. The earlier timing pulse, TPA may also be used when a general purpose strobe is required.

Memory timing requirements for user-added memory have been given earlier in this Manual under **Memory Addressing and Expansion.**

Troubleshooting

After the Development System is plugged in and POWER switch turned on, the display lights should come on.

Depressing RESET should cause the RUN indicator to turn off. If it does not, noise or extraneous signals may be present on any of the DMA-OUT, DMA-IN, or INTERRUPT lines.

Depressing RUN U should cause the RUN indicator to light. If it does not, the problem may be a failure in the control module, no CLOCK signal, a burned-out indicator, or a failed CPU.

Another common problem is data bus contention caused by enabling user-inserted I/O devices or memory onto the data bus at the wrong time. In particular, if extra memory modules have been added, check that they are wired to the correct Memory Bank Select signal.

So long as CLOCK and dc power (V_{DD} and CPUPWR) are present at the CPU module, then TPA and TPB should also be present unless CLEAR-N or WAIT-N is asserted.

Pressing LF or CR after RUNU should cause UT20 to calibrate itself and type out a prompting asterisk. If the CDS does not respond to UT20 commands, then locating the source of trouble is beyond the scope of this Manual. In particular, the troubleshooting of user-designed interfaces is an art.

Hardware Specifications

NEST

19" rack mount, 5.25" high, 10" deep.
32 card positions (7 occupied by power supply).
Connectors with plastic guides. ●

44 pins; 0.156" pin spacing; wire-wrap pins
0.5" connector spacing

PANEL

Hinged at left; knob provided to latch panel.

Seven switches

POWER ON
RESET
RUN P
RUN U
LOAD
SINGLE STEP or CONTINUOUS
BUS or LAST I/O BYTE DISPLAY

Six Hex Digit Displays for MEM ADD and BUS/I/O

Six LED Indicators:

RUN
Q
SC0
SC1
WAIT
CLEAR

Line cord and socket at back of cage.

POWER SUPPLY

Mounted to slotted rear of nest, uses space of 7 connectors.
+5 volts at 6.0 amperes; 5% regulation
-5 volts at 0.5 ampere; 5% regulation
+12 volts at 0.5 ampere; 5% regulation
Short circuit and thermal protection.
One fuse for AC, on front panel.
(no overvoltage protection)

CABLING

AC power cord (8 feet)

Power supply to Panel:

+5 volts for lamp, ground

Power to control module connector:

Flat cable, 50-pin for interfacing display logic and control switches

Terminal module to (customer furnished) TTY:

Six-wire cable (15 feet)
terminated with MOLEX connectors for TTY

Cable from terminal module to terminal using EIA interface:

15 feet of six-wire cable to 25-pin Cinch plug.

● These connections have unusual sized pins– .015" x .041". For wiring, equipment such as OK Machine (Bronx, N.Y.) electric-powered wire-wrapping tool Model EW 7D or Model G-100 with bit WB2644M and sleeve P3032LN, or equivalent, should be used. Cards inserted in these connectors should have beveled edges to avoid deforming the contacts.

CDS Resident Software Development Aids

A resident software aid is a program which runs on the RCA COSMAC Development System, is stored in or loaded into one of the system's memories, and performs some general function for the user. The Utility Program UT20, for example, is a permanently resident software aid. A program is **permanently** resident when it is stored in ROM, occupying some fixed portion of the addressable memory space. It is **temporarily** resident when it is loaded into some portion of the existing RAM space. One of the fundamental purposes of UT20 is to facilitate this loading process.

In this section of this Manual, two resident software aids are described. These aids are the **COSMAC Resident Assembler** and the **COSMAC Resident Editor**. The COSMAC Resident Assembler translates a program in assembly language into hexadecimal code ready for machine operation. The COSMAC Resident Editor is a programming tool that helps in program creation, correction, and change. It operates interactively with the user at a terminal. This section will provide some useful introductory material that applies specifically to the Assembler and the Editor programs.

CDS I/O Terminals

Both the Assembler and the Editor programs process an input or source file and produce an output file. For purposes of this Manual, a **file** may be considered to be a sequence of records or lines (each consisting of a sequence of **characters**) stored in some storage medium. The program reads the input file, processes it in some way, and writes an output file.

Three versions of the Resident Assembler and Editor are available for use with the CDS. One is a paper-tape version of both programs for use with a **Teletype** (TTY) terminal. The second is a magnetic cartridge version supplied for the Texas Instrument "Silent 700" terminal or equivalent. In this manual, the **TI Model 733 ASR with "Remote Device Control"** option is assumed. A third version of the resident software is supplied on a diskette to purchasers of the CDS Floppy Disk option (CDP18S805). With this option, any standard data terminal can be used that will interface to the CDS i.e., having a serial ASCII 20-mA or RS232 interface with a baud rate of 110, 300 or 1200 baud. The **Floppy Disk Manaul MPM-217** describes the use of the resident software on that system. The discussion in this manual concerns only the paper tape and magnetic cartridge versions. If a Teletype terminal is used (with local files on paper tape), it must be additionally outfitted with a Remote Reader Control feature (see Appendix C) to permit the running program to start and stop the paper-tape reader.

While the resident program runs, the terminal device (Teletype or TI terminal) should operate in the line mode, with media (paper tapes or magnetic tape cassettes) properly mounted and with the manual media control switches properly set. As before, the terminals are to be in the **full duplex** mode and set for the appropriate baud rate.

Memory Space Requirements

In addition to the memory area occupied by the resident program, RAM "work" space is normally required. This work space can be used for many purposes. For example, typically it contains an input buffer area into which one or more input lines are read from the source file. An output buffer may also be included into which data is accumulated prior to writing to the output file. Space may also be required to build data tables whose contents depend on information in the input file.

The COSMAC Resident Assembler and Editor are read-only programs, i.e., any memory writes which take place during execution occur in the work space.

The program itself resides on some input file (paper tape or magnetic tape cassette), and is loaded into RAM by use of standard UT20 loading techniques previously described (the "!M" command). Once the program is loaded, control is transferred to it, again by the standard UT20 command $U(CR). Once it is in control (running), it proceeds to communicate with the user via the I/O terminal, outputting its own prompt messages on the printer, reading user commands from the keyboard, and appropriately processing the input file to generate an output file.

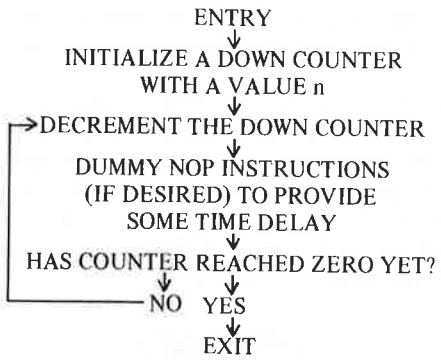
In subsequent operating instructions for the Assembler or the Editor, program space requirements and the minimum amount of work space required (RAM) are defined. If more RAM space is available, the program is designed to take advantage of it by providing some form of "better" service (as explained further later). Both programs load starting at address 0000 and thus cannot be resident simultaneously in the CDS.

Informal Introduction to the COSMAC Resident Assembler

Early in this Manual under the head "Machine Language Programming," a simple time-out test program was discussed. This program in UT20-compatible hexadecimal load form is given by:

!M0 F8FFB1219191913A030Q (CR)

This program was generated using the following flow chart:



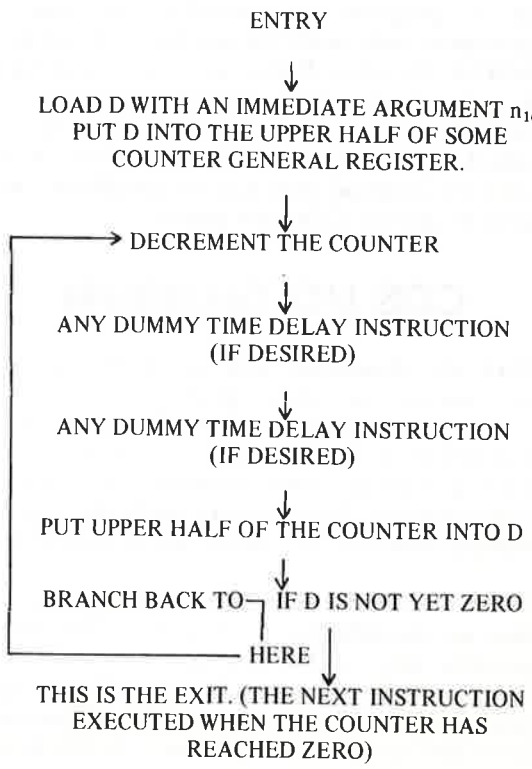
This flow chart, of course, is a much more understandable version of the program. The time from ENTRY to EXIT is approximately n_{16} times the time for one pass through the loop.

An assembly language is designed to permit a machine-readable form of a program whose content is intermediate between that of an English language flow chart, which is easily understood, and that of a machine language hexadecimal string, which is essentially impossible to "read". A proper assembly language program, containing mostly English-like text, can be directly "read" and understood. An assembler is a program which converts the assembly language version into its equivalent machine language form.

Flow Chart to Operation Mnemonics

The time-out text program given above can be used to illustrate some of the essential properties of the COSMAC Resident Assembler starting from the flow chart and proceeding toward the hex form "by hand."

A next version of the program, expressed in terms of specific COSMAC instructions, is shown below:



The use of short hand mnemonics for the instructions and appending comments gives:

```
        LDI n          .. n IS APPROXIMATELY THE
                       .. NUMBER
        PHI COUNTER    .. OF 256 LOOP PASSES BELOW
->      DEC COUNTER    .. REDUCE NUMBER OF PASSES
                       .. REMAINING
        DUMMY          .. JUST TO WASTE TIME
        DUMMY          .. WASTE MORE TIME
        GHI COUNTER    .. SEE IF COUNT HAS YET
                       .. REACHED
        BNZ -+         .. ZERO. LOOP IF NOT
          HERE
        EXIT INSTR     .. TIME EXPIRED. GO ON
```

where LDI, PHI, DEC, GHI, and BNZ are operation mnemonics standing for LOAD IMMEDIATE, PUT HIGH, DECREMENT, GET HIGH, and BRANCH IF NOT ZERO, respectively. Their equivalent hexadecimal codes (for example, 3A for BNZ) can be found in Appendix E and in the **User Manual for the RCA CDP1802 COSMAC Microprocessor,** MPM-201. Each line is now beginning to resemble an assembly language statement.

The last version illustrates two fundamental properties of an assembly language - the use of operation mnemonics and the use of comments. An assembler is designed to recognize operation mnemonics, which are much more descriptive to the programmer, and to convert them into their hexadecimal code equivalents. In addition, an assembler is designed to ignore comment text fields in statements when it recognizes their existence. In the program version above, every comment begins with a double period (..) and extends to the end of the line. Comments are invaluable to the programmer because they permit him to add documentation to a program's statements.

Addressing

The next problem considered is that of assigning **addresses** - specifically, the branch address in the last instruction in the loop. Clearly, addresses which are assigned depend on where the program will reside in memory while it is executing. If it is assumed that this location is not presently known absolutely (for example, because the routine's exact location within a larger program may change), a labelling procedure may be defined to replace the arrowed path shown. Two examples are given below:

Example 1) LABEL: DEC COUNTER

.

.

.

BNZ LABEL

An assembler permits locations within a program to be identified by English-like **symbols** (e.g., "LABEL:" above). Then any reference to a location may be made by use of its label (e.g., "BNZ LABEL"). The programmer is free to select almost any sequence of up to 6 characters for each label. Typically, he chooses a symbol which has some logical meaning within the context of his program (e.g., LOOP, DELAY, TEST1, SEARCH, etc.). During the process of translating the program's statements, the assembler keeps track of the addresses of all bytes it generates (starting from some known address reference, such as zero). It uses an internal **location counter** for this purpose. Whenever it encounters a LABELed statement, it enters the address of the instruction in a **symbol table.** All references to that label may then be replaced with appropriate address bytes.

Example 2) BNZ *-m

An assembler also normally permits addressing relative to the position at which the reference is found. The special symbol "*" is meant to refer to the address of this statement and m is a count of the number of bytes from this point.

Two forms of **symbolic addressing** have been defined: using a statement label or using the symbol "*". One form of the program now becomes:

```
BEGIN: LDI n          .. n IS APPROXIMATELY THE
                      .. NUMBER
       PHI COUNT      .. OF 256 LOOP PASSES BELOW
LOOP:  DEC COUNT      .. REDUCE NUMBER OF
                      .. PASSES REMAINING
       DUMMY          .. JUST TO WASTE TIME
       DUMMY          .. WASTE MORE TIME
       GHI COUNT      .. SEE IF COUNT HAS REACHED
       BNZ LOOP       .. ZERO. LOOP IF NOT
EXIT:  IDL            .. STOP AFTER TIME DELAY
                      .. HAS EXPIRED
```

which is almost a correct assembly language program. (Notice that three statement labels have been specified. Only one is presently referenced).

Assembly Language Equivalent

Next to be discussed are the selection of the value for n, the selection of the COUNT register, and the "DUMMY" instruction. To get the maximum delay, the original version of this program used a hex FF for the immediate byte. The assembly language statement LDI #FF will translate properly. This selection provides an example of the fact that there are still many places in a program where explicit values are specified by the programmer. The "#" indicates the presence of an explicit hex constant. One can similarly explicitly identify the general register to be used as the counter with statements such as PHI 1, DEC 1, etc., assuming R1 was chosen. Suppose, however, that one wished to defer or later modify register assignments. A convenient permissible procedure is to continue to use the symbol as an identifier (in this case not of a memory location but of a general register) and to give the symbol a value with a special statement called an EQUATE statement, which has the form COUNT=1. In this case, all occurrences of COUNT will be replaced with 1 by the assembler. If, later, one wished to reassign registers, a change to COUNT=10, for example, would automatically change all references to COUNT to hex value #0A.

To generate a delay, one may use the NOP instruction or any other time-wasting instruction. The hex program originally given merely repeated the GHI 1 instruction three times. There are several ways by which this instruction can be expressed to the assembler. One in particular uses another form of EQUATE statement to give a value to a symbol. As will be explained later, a comma may be used to precede many kinds of "constants", some whose values are explicitly stated and some whose values are derived by the assembler. In particular, the statement "DUMMY", for example, will cause a substitution of the value for the symbol. Thus, if another statement DUMMY=#91 is supplied, a means is again provided by which all occurrences of DUMMY will be replaced by a hex 91 (which is a GHI 1 instruction).

Finally, the assembler begins assigning address values starting with zero. A special statement is provided to cause the assembler to change the present value in its internal location counter if required. It is called an ORG statement. The final form of one assembly language equivalent of the hex program started with is then:

```
BEGIN:  LDI #FF          . .INITIALIZE COUNTER
                         . .REGISTER FOR
        PHI COUNT        . .ABOUT 65000 PASSES.
LOOP:   DEC COUNT        . .REDUCE # PASSES
                         . .REMAINING BY 1.
        ,DUMMY           . .JUST TO WASTE TIME.
        ,DUMMY           . .WASTE MORE TIME.
        GHI COUNT        . .SEE IF COUNT HAS
                         . .YET REACHED
        BNZ LOOP         . .ZERO. LOOP IF NOT.
EXIT:   IDL              . .STOP AFTER TIME DELAY
                         . .HAS EXPIRED
        COUNT = 1        . .REGISTER 1 ASSIGNED AS
                         . .THE COUNTER
        DUMMY = #91      . .NOP IS A REPEAT OF A
                         . .GHI 1 INSTRUCTION.
        END              . .REQUIRED LAST STATE-
                         . .MENT IN EVERY PROGRAM.
```

COSMAC Resident Assembler

Assembler Operation

The COSMAC Resident Assembler (CRA) is a program which provides for assembly of COSMAC programs without the use of another computer. CRA runs directly on the COSMAC Development System itself in a stand-alone manner. It converts source programs written in COSMAC Level I Assembly Language into executable (hexadecimal) machine code.

The use of an assembler permits the programmer to write programs using convenient symbols and expressions. The input or source program consists of a sequence of statements. A statement is normally translated by the assembler into an equivalent sequence of hexadecimal digits (a single machine instruction or a data field of user-defined constants). This code is then placed in its proper position (i.e., assembled) in an output or **object** file - which is the executable machine program. Some statements are special control commands to the assembler. They are called assembler **directives**. They are distinguished by the fact that they do not directly cause output code to be produced.

An assembly language program (as compared to its machine code counterpart) is easier to write and to understand. Each statement may be annotated with user comments which are ignored by the assembler

but carried along for documentation purposes. The assembly language program is easier to change and contributes to fewer trivial errors than a corresponding machine code program.

The Location Counter

The basic function of CRA is to fill a simulated COSMAC memory (the object code area or file) with the hexadecimal equivalent of the user's source program. For this purpose, CRA maintains a two-byte location counter as a pointer into this area. The initial value of the location counter is zero. As CRA runs and produces output code, it places this code in the output area at the position specified by the location counter, and then advances the location counter past the bytes just inserted. The value of the location counter is also controlled by the CRA directive statements ORG and PAGE (explained later). These statements may be used, for example, to advance the location counter past an area without filling it.

The source programmer may explicitly refer to the current location counter value by use of the character "*" in an expression (see later discussion).

The Symbol Table

The most fundamental logical facility provided by an assembler to aid in the output code generating process is its ability to construct and reference a symbol table. **A symbol is a sequence of one to six alphanumeric characters beginning with a letter.**. Each programmer-defined symbol is given an entry in this table where it is assigned a two-byte value which is often an address value equal to the location counter contents at the point where the symbol is "defined" in the program. However, it may also be a user-specified value (for example, a constant) if the symbol is defined in an EQUATE directive (explained later). Thus, while a symbol most often represents an address, it may also represent a specific CPU register, an I/O device number, or any other user-specified value (e.g., a constant, an immediate argument, etc.).

A symbol normally appears at many points in the source program. At one of these points, it is assigned a value in the symbol table; i.e., it is defined. At all other points (whenever the symbol is referenced), this value is used by the assembler to derive or produce code. Thus, by changing the value assigned to a single symbol, the programmer can make substantial changes in his object code file.

A symbol is also called a label, a name, an identifier, or a symbolic address or pointer.

Expression Evaluation

As CRA processes source statements, it produces hexadecimal code values. Much of this code is the direct equivalent of explicit, user-specified information in the source program (constants, register or device numbers, operation code mnemonics). Other code values are derived indirectly, using either the current value in the location counter or the value assigned to a specific symbol in the symbol table. The code values thus produced are either assembled into the output stream, as previously described, or assigned to new symbol table entries (when new symbols are defined).

At various places in the source program, CRA will be expecting to encounter an expression. **An expression is defined as one of the following forms:**

1) expression constant	#2F or 47
2) *	*
3) *±expression constant	*+47 or *–#F1
4) symbol	SAM or J65MP
5) symbol ± expression constant	AREA9–#2F

where an expression constant contains an explicit hexadecimal or decimal value. (The acceptable forms for expression constants are described later.) CRA evaluates the expression by using the explicit constant value (if present), the current location counter value (for the "*", if it is present), the symbol's value in the symbol table (if a symbol is present), and by performing the required arithmetic operation (if necessary). The result is always a two-byte value which may be disposed of in any one of several ways, as discussed further later. Spaces adjacent to the + or - operators are optional.

Following is a more detailed definition of the syntax which CRA is designed to recognize.

COSMAC Level I Assembly Language

Lines and Comments ▲

Each line or record in the source file is distinguished by an ending carriage return character. A line may consist only of a comment or of one or more statements optionally followed by a comment.

▲ NOTE: All discussion regarding special CRA punctuation characters (such as semicolon, colon, period, asterisk, parenthesis, equals sign, number sign, apostrophe, etc.) refer to those which do not appear within text constants (defined later). Any character within a text constant has no special punctuation significance to CRA.

A comment is any series of characters beginning with two periods. It extends to the end of the line. Thus, the occurrence of two periods at any point in a line causes CRA to ignore the remainder of the line. Statements within a line are normally separated by semicolons (with the last statement optionally terminated by a semicolon). Within each statement, spaces (blanks) may be used freely (except within symbolics and mnemonics) in order to improve readability. CRA will ignore them.

In all the examples which follow, a pair of square brackets will be used to enclose an **optional** entity - one which may or may not be included. Examples of valid lines are then:

1) . . COMMENT
2) $STATEMENT_1$ [; $STATEMENT_2$;· · ·; $STATEMENT_n$] [;] [. . COMMENT]

Symbol Definitions

(Statement Labels and Equate Statements)

Any statement may optionally begin with a symbol (called a "statement label") immediately followed by a colon. Under these conditions, the symbol is entered into the symbol table and assigned the present location counter value. A statement thus has the form:

[SYMBOL:] STATEMENT BODY

(For example, LOOP: INC R4)

A symbol is also defined when it appears as the left-hand part of an EQUATE statement, which has the form:

SYMBOL=EXPRESSION

(For example, READER=6)

In this case, the expression is evaluated and the resulting two-byte value is assigned to the symbol in the symbol table. (Acceptable forms for symbols and expressions have already been explained.)

Thus, a symbol definition is indicated to CRA by the occurrence of ":" or "=" immediately after a leading sequence of alphanumeric characters in a statement.

When equating a symbol to a register number, only a decimal or a hexadecimal number should appear on the right side of the equation. For example:

COUNTR= 7 is correct
COUNTR= #07 is correct
COUNTR= R7 is incorrect

DELAY= COUNTR is also correct

Explicit Constants

At numerous points in the source program, the programmer desires to directly specify explicit constants to CRA. Most often (but not always) the hexadecimal equivalent of an explicit constant is inserted directly into the output code stream at the point where it appears in the source program. (For example, initial data values and immediate arguments may be explicitly defined this way.) CRA allows the programmer the ability to specify absolute constants in binary, hexadecimal, decimal, and alphanumeric forms. The possible explicit constants are summarized below.

Hexadecimal constants: A hex constant is specified with either of the following forms:

	Example
1) #hh...hh	#3E0F
2) X'hh..hh'	X'3E0F'

where each h is a hex digit (0 to F). **CRA requires that an even number of hex digits be specified.** There are further restrictions on hex constant lengths under certain conditions.

Decimal constants: A decimal constant is specified with either of the following forms:

	Example
1) dd...dd	635
2) F'dd...dd'	D'635'

where each d is a decimal digit (0 to 9). Each such constant is converted into hex, producing one or two bytes, depending on the space required to represent it. Decimal values greater than 65535 are converted to hex but then truncated to two bytes (upper bytes removed).

Expression constants: An expression constant may be either form of the hex constant or the first form of the decimal constant. Because an expression translates to two bytes, a hex expression constant should normally be restricted to two or four digits in length.

Binary constants: A binary constant is specified in the form:

B'bb...bb'

(For example, B'01101')

where each b is 0 or 1. Up to eight bits may be specified. Each such constant is converted to one byte, with leading 0's assumed.

Text constants: A text constant is specified using the form:

T'cc...cc'

(For example, T'THIS IS TEXT')

where each c is any printable character, including space. Each character is converted to its ASCII code equivalent (see Appendix F) and is represented in one byte. Characters that have no graphic associated with them (i.e., ETX, DC-3, CR, LF, etc) should not be used within a text constant. Entering an apostrophe within a text constant is treated specially, however. See "Additional Notes" below. Refer to Example 4 under "Examples of UT20 Read and Type Usage" to see how CR, LF is handled.

Address Constants

The programmer finds it useful to specify not only explicit or absolute constants, but also derived constants whose values are assigned or "computed" by the assembler. Because the fundamental function of the assembler is to assign address values, such constants are normally called address constants. For CRA, an address constant has one of the following forms:

	Example:
1) A(expression)	A(GEORGE + 2)
2) A.1(expression)	A.1(LOOP)
3) A.0(expression)	A.0(*-X'10')

where the permissible forms for an expression have already been defined. For all cases, the resulting constant is derived by first evaluating the expression. In the first case, the two-byte result is the constant. In the second case, only the upper byte is used; for the third case, only the lower byte. For all cases, the resulting one- or two-byte value is assembled directly into the code output.

Operation Mnemonics

CRA uses special two-, three-, and four-character mnemonics to represent the various instructions in the COSMAC instruction set. These mnemonics are listed in Appendix E. When CRA determines that an operation is being specified, it looks it up in a table to determine the code equivalent of the mnemonic. (Note that this table is **not** the symbol table, which contains only programmer-defined symbols.) Thus, use of an operation mnemonic effectively defines an explicit hex code value to be inserted into the object stream.

Instructions and Operands

There are two types of output code-producing statements: instructions and data lists. An **instruction** begins with an instruction operation mnemonic. In some cases (such as IDL, RET, LDX, etc.) this mnemonic is all that needs to be specified. In most cases, however, the operation mnemonic must be followed by an **operand**. The form of the operand (i.e., the additional information which the programmer needs to supply to fully define the instruction) depends on the type of instruction. The four operand forms follow.

Register operands: Many instructions (e.g., INC, LDA, etc.) include a hex digit identifying one of the scratchpad registers. The operand field in such a statement may include either a single hex digit, or a symbol. For the last case, CRA uses the least significant hex digit of the symbol's value in the symbol table as the register identifying field.

Examples:

```
DEC 9

LDA RF

PLO SAVE
```

("SAVE=#0F" could have previously defined SAVE.)

I/O device operands: The instructions OUT and INP require a device-identifying field. The operand in such a statement may be a single digit in the range 1 to 7, or a symbol. Again, for the latter case, a symbol table lookup occurs, using the least significant hex digit of the symbol's value (checking also that it is within the appropriate range).

Examples:

```
OUT 4

INP READER
```

Branch addresses: Every branch instruction requires an operand specifying the branch address. If the mnemonic is a short branch, a one-byte operand is generated. A two-byte operand is generated if the mnemonic is a long branch. Whenever CRA sees a branch operation mnemonic, it expects to next find an operand in the form of an expression. The acceptable forms for expressions have already been defined. In case of a short branch, CRA evaluates the expression by getting a two-byte address, checks that this address is within the current 256-byte page by examining the upper byte, and uses the lower byte as the second byte in the instruction. For a long branch, the upper byte represents the page number, and the lower byte is the address within that page.

Examples:

If A(LABEL) is #6789

BZ LABEL

generates #3289

and LBZ LABEL

generates #C26789

Immediate operands: Several instructions include a second byte as an immediate argument. The operand field in such a statement may be any one-byte constant (i.e., an absolute or explicit constant or an address constant) or a symbol. For the latter case, CRA uses the least significant byte of the symbol's assigned value.

Examples:

XRI X'FF'

ADI INCREM

LDI A.0(*)

Note: When an immediate argument is specified, it is the programmer's responsibility to make sure that it is a one-byte constant. If it is longer, CRA will not generate an error message, but will merely insert the entire constant into the output stream, possibly causing an error during program execution.

Data Lists

The typical program normally includes memory areas which contain data values. Statements which define initial data values are also code-producing statements (although the code generated is normally not "executable"). The data list is a special statement provided for these purposes. It begins with either a comma or the special mnemonic "DC" (which stands for "Define Constant") and is followed by a sequence of one or more constants separated by commas. Each constant may be an absolute, explicit constant (hexadecimal, binary, decimal, or text) or an address constant or a symbol. For the last case, to be consistent with the treatment of symbols as immediate data, CRA substitutes the lower byte of the symbol's assigned value. Thus, a constant in a data list is similar to an immediate operand, but now a length greater than one byte is entirely justifiable.

Examples:

DC X'ABCD',355

,#ABCDEF,T'TEXT',B'011'

(**Note:** Any statement may be directly followed by a data list without the intervening semicolon. For example LDA 9,#3001.)

CRA Directives

The EQUATE DIRECTIVE (of the form SYMBOL=EXPRESSION) has already been discussed. Three other directive statements are also recognized by CRA:

ORG Statement: This statement is written "ORG" followed by an expression. CRA executes this directive by setting the location counter equal to the value of the expression.

Example: ORG *+20 ..Reserve 20_{10} bytes of space

PAGE statement: The PAGE directive, simply written "PAGE", increases the value of the location counter to that of the beginning of the next 256-byte page; i.e., the upper byte of the location counter is incremented and the lower byte is set equal to zero.

END statement: The END directive, written "END", informs CRA to terminate the assembly. It should appear only once, as the last statement in the source program. The END directive is normally followed by a DC3 character. The DC3 is produced by the EDITOR to signify an end of file.

Thus, in addition to recognizing all the instruction operation mnemonics listed in Appendix G, CRA also recognizes the special mnemonics "DC", "ORG", "PAGE", and "END".

Additional Notes

1) As noted earlier, a space is not permitted within a syntactic entity (symbol, mnemonic, constant, etc.). A space is not permitted between a

symbol being defined and the following colon or equals sign. Note, however, that a space within a text constant is permitted. It is translated into its ASCII equivalent code. There is a case where a space is required as a punctuation character. In order to distinguish an operation mnemonic (including ORG) from its following operand (if present), CRA expects to find at least one space.

2) An apostrophe may be included within a text constant by preceding it with a "dummy apostrophe". Thus, the string IT's is written as a text constant as

```
,T'IT''S'
```

3) Special control characters (non-printing characters, such as carriage return, line feed, etc.) should not be placed within the quotes of a text constant. Rather, they should be defined by splitting the text constant into two successive text constants, with the intervening control character represented with a hex constant (using its ASCII code). For example:

```
,T'LINE1',#0D0A,T'LINE2'
```

4) Several COSMAC instructions execute by automatically advancing the pointer to an operand byte after processing it. If the pointer to the operand byte is the same as the current program counter (for example, if X=P or if N=P), then the operand byte may be considered an immediate operand (provided an auto-increment occurs). A statement for such an instruction (under the conditions specified) is most conveniently followed by a comma followed by the one-byte immediate constant. This sequence is permissible because any statement may be immediately followed by a data list - omitting the intervening semicolon.

For example, assuming P=0, the sequence SEX 0; OUT 5 ,X'52' outputs the immediate hex constant, #52, to output port and continues.

5) In general, any symbol may be referenced before it is defined in a program (termed a "forward reference"). Only one restriction exists: A symbol on the right-hand side of an EQUATE statement (i.e., in the expression) must have been previously defined.

6) CRA uses the location counter value before a statement is processed as the value for any "*" occurring within the statement. Thus, for example, for BR *+3, the value used for the * is the location where the branch byte (hex 30) will be placed, not one byte past that. Thus, BN1 * will cause a program loop until flag 1 goes true.

Code Examples and Review

Fig. 24 is a hypothetical program designed not to do anything meaningful, but rather to present examples of various acceptable CRA statements. It contains a listing of the program and the corresponding output code generated. Fig. 25 contains the symbol table for the program. Both were generated by a typical CRA assembly run.

In Fig 24, the left-hand column gives the location counter value before the line was processed. The next column give the output hex code generated at that location by the line. (Terminating semicolons in this column should be ignored. They are present to format the output file properly for subsequent loading of the object program. See later operating instructions.) The next column gives a source program line number for reference purposes, and finally the source code is reproduced. The running comments in the source program refer to the statement examples where they appear.

By reading the source program in detail (paying special attention to the running comments), one can quickly review much of what has been said concerning COSMAC Level I Assembly Language. Output code values may be verified by referring to Appendices E and F. In particular the reader should verify the values assigned to the symbols in Fig. 25.

Error Messages

Whenever CRA detects a violation of its syntax rules, it generates an error message. There are, however, some possible program errors which will not be detected by CRA because they do not result in syntax rule violations. For example, R3=8; INC R3. R3 is now a symbol, the value of which is 8, so register 8 gets incremented.

When there is a syntactical error, CRA indicates it first, by printing the line in violation using its standard listing format (location counter, output code, line sequence number, source line); second, by inserting a "?" at the detection point in the source line; and third, by printing an error code on the next line. If the error is detected at the end of the line, the "?" may be omitted. In most cases, by looking up the error code meaning in the listing which follows and by noting the position of the inserted "?", the user can easily determine the nature of the error.

It should be emphasized, however, that it is possible that an error at one point in a source line may be interpreted by CRA as an error at a different point. For example, in T'TEXT...COMMENT, a single quote is missing after TEXT. It will not be detected until the end of the line. (In fact, if the

```
LOCATION  OUTPUT    LINE
COUNTER   CODE      NUMBER   SOURCE LINES
0000      ;         0001     . . THIS PROGRAM IS NOT DESIGNED TO DO
0000      ;         0002     . . ANYTHING MEANINGFUL. RATHER, IT IS
0000      ;         0003     . . DESIGNED TO ACT AS A REVIEW AND
0000      ;         0004     . . SUMMARY OF THE COSMAC LEVEL 1
0000      ;         0005     . . ASSEMBLY LANGUAGE.
0000      ;         0006     . . A LINE MAY CONSIST ONLY OF A COMMENT
0000      0078F0;   0007     IDL;SAV;LDX          . . MULTIPLE STATEMENTS ON LINE
0003      ;         0008     CRLF = #0D0A         . . SYMBOL DEFINED VIA EQUATE
0003      ;         0009     TALLY = 9            . . SAME, USING DECIMAL CONSTANT
0003      ;         0010     PTR =   TALLY + 1    . . EXPRESSION CONTAINS SYMBOL
0003      29;       0011     LOOP:   DEC TALLY    . . SYMBOL DEFINED VIA LABEL
0004      89;       0012             GLO TALLY   . . TALLY IDENTIFIES A REGISTER
0005      3A03;     0013             BNZ LOOP    . . EXPR CONTAINS SYMBOL ONLY
0007      3016;     0014             BR SAM1     . . A FORWARD REFERENCE
0009      ABCD400100; 0015   ,X'ABCD',64,D'256'  . . SOME CONSTANTS
000E      0354455854; 0016   DC B'011',T'TEXT'   . . MORE CONSTANTS
0013      00120A;   0017     ,A(* - 1),A.0(CRLF)  . . SOME ADDRESS CONSTANTS
0016      1A;       0018     SAM1:   INC A        . . REG NAMED WITH HEX DIGIT
0017      2B;       0019             DEC RB      . . REG NAMED WITH "R" FORM
0018      67;       0020             OUT 7       . . EXPLICIT DEVICE NUMBER
0019      6F;       0021             INP READER  . . DEVICE IDENTIFIER IS SYMBOL
001A      ;         0022     READER = 7           . . ANOTHER FORWARD REFERENCE
001A      F841;     0023             LDI T'A'    . . 1-BYTE IMMEDIATE CONSTANT
001C      FC05;     0024             ADI NUMBER  . . SYMBOLIC IMMEDIATE CONSTANT
001E      ;         0025     NUMBER = X'05'       . . EVEN NUMBER OF HEX DIGITS
001E      3600;     0026             B3 LOOP-#03 . . BR ADDRESS IS EXPRESSION
0020      3C22;     0027             BN1 * + 2   . . EXPR REFERS TO LOC COUNTER
0022      ;         0028             ORG * + #10 . . ADVANCE LOC CTR 16 BYTES
0032      ;         0029             PAGE        . . ADV LOC CTR TO NEXT PAGE
0100      6710;     0030             OUT 7,16    . . IF X = P, #10 IS DATA OUT
0102      4205;     0031             LDA 2,NUMBER . . IF P = 2, SAME AS LDI #05
0104      ;         0032     F = 2    . . BAD PRACTICE TO REDEFINE A HEX CHAR SINCE
0104      12;       0033     INC F    . . THIS WILL INCREMENT R2, NOT RF.
0105      ;         0034     LABEL: ORG *         . . AN EASILY-MOVEABLE LABEL
0105      78;       0035     RET:    SAV;         . . "RET" IS A LABEL HERE
0106      7078;     0036     RET;    SAV;         . . "RET" IS AN INSTRUCTION
0108      F801;     0037     LDI A.1 (LABEL)      . . A PROPER 1-BYTE IMMEDIATE
010A      F80105;   0038     LDI A(LABEL)         . . ERROR. NO DIAGNOSTIC GIVEN
010D      F805;     0039     LDI LABEL            . . LOWER BYTE USED ONLY
010F      1A;       0040     INC CRLF             . . LOWEST HEX DIGIT USED ONLY
0110      0A;       0041     ,CRLF                . . LOWEST BYTE USED ONLY
0111      ;         0042     END                  . . REQUIRED LAST STATEMENT
0000
```

Fig. 24 — CRA listing of sample program.

Hex Value	Symbol
0D0A	CRLF
0009	TALLY
000A	PTR
0003	LOOP
0016	SAM1
0007	READER
0005	NUMBER
0002	F
0105	LABEL
0105	RET
0000	

Fig. 25 — Symbol table for sample program of Fig. 24.

comment happens to end in a single quote, the error will go undetected.) Further, and more important, it is possible for the error code to indicate one type of error when another actually occurred. For example, the statement SAM INC 3 is missing a colon after the label SAM. The primary meaning of the error code which will return in this case is: "unrecognized mnemonic". This response is understandable because if CRA does not detect a colon or an equals sign, it assumes that the statement does not begin with a symbol. If therefore expects a mnemonic or a comma and does not find either.

An error on one line may cause several lines to be flagged. This response typically occurs when a line containing a label is flagged because of a missing colon and all subsequent references to that label are also noted as "undefined."

Whenever an error exists, the output code is questionable. However, as best it can, CRA increments its location counter past this code and continues to process the source program, possibly detecting further errors which it flags similarly. Detection of an error does not stop assembly of a program. CRA continues in its attempt to find all syntax errors.

The possible error codes and their meanings are given in Table VII, and a summary of error messages is given in Table VIII. If in the process of generating a listing CRA creates a line that exceeds the standard length (typically 78 characters), the line is broken by a (CR) (LF) sequence. The rest of the line is continued on the next line but is preceded by a continuation mark - a period.

TABLE VII – CRA ERROR CODES AND THEIR MEANINGS

Error Code	Meaning
01	UNRECOGNIZED MNEMONIC OR MISSING COMMA The body of a statement (other than EQUATE) must begin with either a valid operation mnemonic or DC, ORG, PAGE, END, or a comma.
02	PREVIOUSLY DEFINED SYMBOL An attempt has been made to define a symbol which already has an entry (and a value) into the symbol table.
04	INVALID CHARACTER WITHIN BINARY CONSTANT CRA is in the process of evaluating a binary constant and has found a character other than 0/1 or the trailing single quote (which may be missing).
05	BINARY CONSTANT TOO LONG The limit is eight bits.
06	EXPECTED HEX OR DECIMAL CONSTANT HERE CRA is in the process of evaluating an expression or a constant and expects to see a hex or decimal constant at this point and does not find one. (Note: Under certain conditions, this diagnostic may occur as the result of an undefined symbol.)
07	UNDEFINED SYMBOL CRA encounters a symbol reference and wants to use its value, but does not find it listed in the symbol table.
08	EXPECTED EXPRESSION HERE CRA determined that an expression was to follow next and did not find leading characters which were proper.
09	INVALID CHARACTER WITHIN HEX CONSTANT CRA is in the process of evaluating a hex constant and has found an invalid character. (This error code may be caused by an uneven number of hex digits.)
10	MISSING TRAILING QUOTE IN TEXT CONSTANT Note that the error marker "?" will not appear because this error is always detected at the end of a line.
11	PERIOD ERROR Either illegal use of a single period or a missing period beginning a comment.
12	LEADING CHARACTER ERROR At the beginning of a statement, a leading alphabetic or comma was not found.

(Cont'd)

TABLE VII – Cont'd

14	BRANCH OUT OF PAGE A branch address was evaluated and the upper byte did not agree with that of the location counter (see Note 1).
15	INVALID REGISTER NUMBER The LDN R0 is an illegal operation. It would otherwise assemble into a hex 00 operation code, the code for an idle instruction.
16	DEVICE NUMBER OUT OF RANGE In an OUT or INP instruction, the explicit or symbolic device number had a value greater than 7.
OVFL	SYMBOL TABLE SPACE EXHAUSTED The amount of RAM allocated to the symbol table has been used up. The user must remove labels by, for example, using *'s more often or branching relative to existing labels. The program can also be broken up into parts, or more RAM can be added to the CDS. (Note: This error condition halts CRA. A restart of the assembler is required, i.e., $U).

Notes

(1) While CRA makes every effort to increment the location counter properly when it processes statements which contain errors, it is, of course, possible for the location counter to have a value at any given point which is different from that which it would have for an error-free program. As a result, it is conceivable that the BRANCH OUT OF PAGE error diagnostic either will occur erroneously or will not be generated when it should be.

(2) The semicolon separating multiple statements in a line is used mainly as a checking device. When CRA has processed an error-free statement and is reinitialized to look at the next one, the intervening semicolon is merely ignored. Thus, it is not really mandatory that the semicolon be used between two statements on a line, if the first is "known" not to contain errors. Clearly, however, it is a necessary practice not only for readability, but also for verification purposes.

TABLE VIII – SUMMARY OF CRA ERROR MESSAGES

Error Code	Meaning
01	UNRECOGNIZED MNEMONIC OR MISSING COMMA
02	PREVIOUSLY DEFINED SYMBOL
04	INVALID CHARACTER WITHIN BINARY CONSTANT
05	BINARY CONSTANT TOO LONG
06	EXPECTED HEX OR DECIMAL CONSTANT HERE
07	UNDEFINED SYMBOL
08	EXPECTED EXPRESSION HERE
09	INVALID CHARACTER WITHIN HEX CONSTANT
10	MISSING TRAILING QUOTE IN TEXT CONSTANT
11	PERIOD ERROR
12	LEADING CHARACTER ERROR
14	BRANCH OUT OF PAGE
15	INVALID REGISTER NUMBER
16	DEVICE NUMBER OUT OF RANGE
OVFL	SYMBOL TABLE SPACE EXHAUSTED

CRA Operating Instructions

Summary of CRA Operating Steps

Before a summary of the detailed operating steps for CRA is given, it should be pointed out that the source file which CRA processes may be derived in any one of several ways. First, it may be written on magnetic or paper tape from the keyboard by operating the terminal in the LOCAL mode. Second, it may already exist (saved) in a remote system; e.g., a time-sharing system. In this case it can be automatically written on a local tape through a communications link, for example. Third, and preferably, it may be generated by use of the COSMAC Resident Editor program, which will be discussed next. In any event, it is assumed that such a source tape exists as a prerequisite to the operating steps listed in Table IX.

TABLE IX – SUMMARY OF CRA OPERATING STEPS

1. Load CRA into RAM using the following steps.
   a. For TI terminal in LINE mode:
      Press RESET, then RUNU, then CR to initialize UT20.
      Mount CRA program cassette.
      Rewind cassette.
      Press LOAD/FF to advance to the first file.
      Press CONT/START to start reading file.
   b. For Teletype terminal in LINE mode:
      Press RESET, then RUNU, then CR to initialize UT20.
      Mount paper tape in reader.
      Place READER CONTROL switch (previously installed) in MANUAL position.
      Press START on the tape reader.
      After completion, set READER CONTROL switch back to REMOTE (program control) position.
      CRA is now loaded into memory.
2. Mount input source. If paper tape, turn reader on. READER CONTROL switch should be in REMOTE position.
3. Mount output object tape. If paper tape, do not turn punch ON yet.
4. Type $U(CR) to transfer control to CRA.
5. CRA begins by typing ?, asking user to identify type terminal in use. In response, type P for "punch" if a Teletype terminal, or any other character if TI terminal.
6. After CRA prompt message (? F, H, L, U =), type F. CRA will then execute its first pass, generating a symbol table and possibly some error diagnostics. If TI terminal is used, source tape will be automatically rewound after completion. If Teletype terminal is used, source tape must be repositioned. After completion, CRA will repeat prompt message.
7. Type L. Listing will ensue. If TI terminal is used, an object tape will be written automatically.
8. Insert this step only to effect Teletype terminal third pass. This pass will punch the object code paper tape. First, turn the paper punch ON. Then reposition the source tape and respond to new CRA prompt message by typing H.
9. After CRA has completed the assembly, and if the object tape is to be run, the user must initialize the object tape for reading. (For the TI terminal, the object tape is already rewound and the user need only change the controls on the terminal so that the just-RECORDED cassette now becomes the PLAYBACK cassette.) Respond to the new CRA prompt by typing U followed by a CR. UT20 will assume control and deliver the * prompt. By following the steps given in 1(a) or 1(b), the object tape can be loaded into memory. Control is transferred by the appropriate $U(CR) command.

RAM Considerations

The CRA program occupies approximately 2.5 kilobytes of memory. It is supplied on UT20-compatible paper tape for TTY use and on cassette for use with the TI terminal.

CRA requires an additional work space of at least 1 kilobyte of RAM for I/O buffers and, particularly, for the symbol table which it constructs and references. Most of the available RAM work space is used for the symbol table. Each entry has a variable length and contains the symbol (one byte per character), a special delimiter control byte, and a two-byte value. Because a symbol is 1 to 6 bytes in length, each symbol entry is 4 to 9 bytes in length. CRA makes use of additional RAM space, if it has been provided, by starting at location 8000, decrementing down, and testing for RAM by successive write/read operations. It stops when the first RAM byte is found and assumes that RAM exists from there down to location 0000. Therefore, **any added RAM must be contiguous** to the original 4 kilobytes supplied.

In 1 kilobyte of work space, there is room for approximately 80 symbol table entries depending, of course, on the average symbol length. The required work space is not a function of the number of statements in the source program being assembled. Rather, it is a function of the number of symbols defined within it.

CRA is a two-pass assembler. That is, normally it reads the complete source file twice to complete an assembly. During the first pass, the symbol table is constructed in RAM and printed on the terminal. Syntactic errors are flagged. On the second pass, object code is generated using the symbol table values just derived and an assembly listing is printed. Addition program errors may be flagged on the second pass. For example, the UNDEFINED SYMBOL error normally occurs here.

Output Options

When the TI terminal is used, the output or object tape is generated automatically during the second pass. This terminal includes means by which information printed on the printer may be different from that which is written on the output tape cassette. In this case, after the second pass, while the printed listing contains all the information discussed earlier (i.e., output code values and associated addresses, and sequence numbers and associated source lines), the output tape contains only output code and address information in a format compatible for subsequent loading via UT20. The output tape file begins with a "!M" message, followed by a sequence of lines which have the UT20 semicolon loading format (where each line begins with an address followed by a sequence of hex digits to be loaded there, terminating with a semicolon).

If the I/O terminal is a teletypewriter, on the other hand, then when the paper tape punch is activated, information punched on the tape is the same as that printed on the printer. Thus, if the punch were turned on at the beginning of a second pass, the entire listing would be written on the output paper tape. The resulting tape could be used for subsequent loading if desired because UT20 ignores any information on an input !M line after the semicolon has occurred. However, this procedure has several disadvantages. The output tape is much longer than it has to be, with most of the information on it extraneous. As a result, the time to read it (during subsequent loading), particularly at 10 characters per second, is normally prohibitive. To handle this problem when a Teletype terminal is used, CRA includes the ability to make two kinds of "second passes" one of which omits the printing of the source program lines and their associated sequence numbers. Thus the normal procedure to generate a paper tape object file is to make a third pass (of the type just described) over the source file to generate an output tape which contains the same information as would be written on a magnetic tape cassette during the second pass.

Thus, several paper-tape, second-pass options exist for the user. First, the normal second pass (generating a listing) may be elected (option L) with the punch off, waiting for the third pass to generate the output tape (option H). Second, particularly for short programs, the punch may be activated for this listing pass - in which case the output tape will be longer than necessary, but a third pass will not be required. Third, particularly if an adequate listing already exists as the result of a previous run, the user may elect the object-code-only option (H) on the second pass, with the punch turned ON.

Prompt Messages

CRA begins any pass with a prompt message which appears as

?F,H,L,U=

asking the user to type one of the letters shown to define what CRA will do next. It is assumed that all I/O media involved - input and output magnetic or paper tapes - are properly mounted. Typing F selects the first pass (symbol table construction). Typing L selects the Listing pass (the normal second pass). Typing H selects the Hex-only listing pass (the normal third pass when paper tape is used). Typing U causes CRA to return control to UT20 (presumably

after the completion of an assembly, in order to load the program just assembled). UT20 receives control and prints an "*" prompt as usual. The user may then re-initialize the object tape for loading and running.

Informal Introduction to the COSMAC Resident Editor

After the user has written his COSMAC assembly language program and wants to assemble and run it, he immediately faces the problem of converting the hand-written source file into a machine-readable form. This conversion involves a keyboard-to-tape operation in which lines on the coding sheet are transcribed to become lines on a source tape. Although an "off-line" process can be executed, in which the I/O terminal is operated in the LOCAL mode, it is much more likely that the COSMAC Resident Editor will be used at this point to create the source tape. The reason is that use of the Editor provides assurance that the created files are in proper format for later reading by the Assembler and for later modification, if necessary, by the Editor. Details on formats are given in the description of the Editor which follows in the next section of this Manual.

Once a source file has been created and a first Assembly run made, it is very likely that error diagnostics will be returned by CRA asking for corrections to the source file to conform to CRA's rules. Typically, the changes required at this point are "trivial" but necessary. For example, spaces may have to be removed in one or more expressions. The same symbol may have been erroneously used for two purposes. An operation mnemonic may have been misspelled or a punctuation character such as a comma, colon, or single quote omitted. The number of possible trivial errors is clearly large.

To correct the errors and to alter the source file to conform the program to the CRA rules, the Editor is used. Typically, modifications at this point merely involve insertion and deletion of single characters or replacement of a small string of characters by a substitute string. The erroneous source file is used as an input to the Editor and the user generates a corrected source file as an output. The new file is then re-assembled. At this point other trivial errors may appear which were not apparent to CRA on the first run. For example, an erroneous instruction operand may not have been flagged on the first assembly because its associated statement label or operation mnemonic may have also been in error. Thus, a new Edit-Reassemble pass may be necessary. Finally, a program is developed to which CRA does not object. At this point, a first run can take place.

The probability of a logical error in the program depends on its length and the previous experience of the programmer. Assuming one or more logical errors are found (via some "debugging" procedure), the source file must again be modified. Often such modifications are no longer trivial. For example, it may be necessary to find all instructions which branch to a given location and precede some of them with one or more instructions currently not in the program. Often, it may be necessary to delete some code or insert some code or move some code to a different point in the program. Several duplicated sets of in-line instructions may have to be removed and replaced with calls to one common subroutine which is to be added. The user may decide to "clean up" the program logically, in any one of several ways, or to improve its "readability" by modifying its comments or statement formats (by inserting TAB's or SPACE's, for example).

Such modifications to the source file also involve use of the Editor. After they are completed, a reassembly may again turn up new errors of the "trivial" variety. And so on. Thus the generation of a bug-free program typically involves the chart shown in Fig. 26. It is thus quite likely that the amount of time spent "conversing" with the Editor will be much larger than that spent with the Assembler.

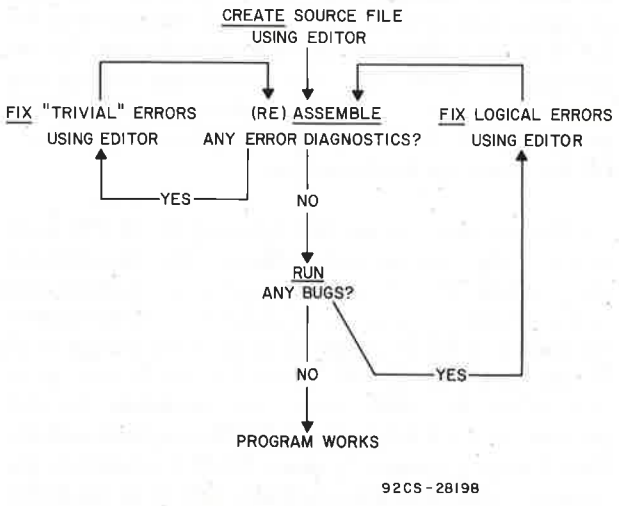


Fig. 26 — Flow chart for "bug-free" program.

A source program may be viewed as a long sequence of **characters**. When the COSMAC Resident Editor reads the source file, it places this character sequence in memory, with the code in each memory byte representing one source program character. The user is then free to type commands to the Editor to manipulate the memory representation of the program. For example, the user may identify a specific location and specify a character sequence to be inserted there. He may also identify certain characters to be deleted or altered. He may ask the Editor to search for the occurrence of specific character sequences, after which further memory modifications (corrections) may be made. (Details of available commands are given later).

After he is satisfied that the new memory representation of the file contains all the desired changes (frequently the user begins an editing session with a hand-written list of the changes to be made), he asks the Editor to write (create) a new file containing the new version of the program. This new file is then used as the input file for a reassembly.

COSMAC Resident Editor

The COSMAC Resident Editor (CRE) is a program which facilitates creation and modification of local COSMAC file. These files are stored on paper tape or on magnetic tape in cassettes. Typically, they are COSMAC source programs. However, they may also be any other kind of conventional document. CRE runs directly on the COSMAC Development System itself in a stand-alone manner. No external computational facilities are required.

CRE Operating Considerations

Memory Space Requirements

The CRE program occupies approximately 2.5 kilobytes of memory space. Like CRA, it is supplied on paper tape and in a cassette for loading into the RAM of the COSMAC Development System. All the information previously given regarding loading and transferring control to CRA applies equally to CRE. See Table IX Summary of CRA Operating Steps in **CRA Operating Instructions**.

CRE requires about 100 bytes of the RAM work space for its own internal purposes. The remainder of the available RAM space is used as an editing area called a **buffer**. Virtually all CRE operations involve the buffer. CRE is designed to take advantage of all of the available RAM space for its buffer area. Approximately 1400 bytes are available for this purpose in the 4 kilobytes of RAM supplied with the Development System. If more RAM is present in the system, CRE will automatically add it to its buffer area. It tests for additional RAM the same way CRA does.

Input and Output Files

Normally, a user creates a file using CRE by filling the buffer from the I/O terminal keyboard and then causing CRE to write this information onto an output tape (which will contain the created file).

An existing (input) file may be modified (edited) by reading portions of it into the buffer, then using CRE commands to alter the contents of the buffer, and finally writing the results onto the output file. Typically, the output file is a new version of the input file. After an editing session, the new version is retained and the old version is discarded (although it may be temporarily saved for future reference or backup).

Thus CRE has means to read an input file into the buffer, means to examine and modify the contents of the buffer in many ways, and means to write the buffer contents onto an output file. Alternatively, when an input file does not exist, the user creates an output file by loading the buffer from the keyboard.

Record Formats

In order to understand the various commands which CRE is designed to execute, it is fundamentally important that the user understand how information is normally recorded on the I/O media (tapes) and in the buffer.

A file is a sequence of records or **lines**. Each line consists of a sequence of **characters**. The length of a line is restricted to 78 or fewer characters of data. Thus, a line in a file is normally printable as a line on the I/O terminal printer. Each character is represented by an 8-bit code or byte, either on the tape or in memory. Typically, every character in a

line is a **printable** character (including space or blank). Every non-printing character code represents a **control** character. A control code may be generated on the keyboard either by hitting an appropriately marked key (e.g., RETURN, ESC, etc.) or by depression of the CTRL button while hitting another key. The printer reacts to the receipt of a control character in one of several possible ways. Some control characters (such as carriage return, line feed, bell, etc.) cause the printer to execute a specific control function. Other control codes either are ignored by the printer or may cause the equivalent of a space on the printed line.

A line in a file may contain control characters (with certain restrictions to be discussed later). If it does, it is quite possible that its printed record will not completely reflect the true contents of the line. CRE treats most control characters which it encounters within a line in the same manner as it treats printing characters. However, certain control characters have special meaning to CRE.

The proper format for tape files is shown in Fig. 27. Each line is terminated with a CARRIAGE RETURN (CR), LINE FEED (LF) pair, followed by a field of six nulls. The NULL character (hex code 00) is ignored by the system. A set of nulls appears after each CR, LF, pair merely to provide a sufficient time delay for the printer carriage to settle to the new line when the tape contents are being printed. Note that the last line on the tape should be followed by a "dummy" line containing only the single data character DC. DC3 is a special control character (generated on the keyboard by hitting CTRL and S). It acts as an END OF FILE indicator. Note also that if the file is stored on paper tape, it is normally preceded by a leader of nulls and followed by a trailer of nulls. The null leader permits arbitrary initial positioning of the tape in the reader.

Tape records read by CRE are deposited into the buffer as they appear on the tape, but with all LF's and NULL's **ignored**. (Incidentally, the DEL or RUBOUT character, hex code 7F, is also ignored on tape input) While CRE operates on the data in its buffer, it specifically uses the CR character as an indicator of the end of a line. (Recall that a line has a variable length.) A new line is assumed to start with the next character in the buffer. The buffer format is shown in Fig. 28.

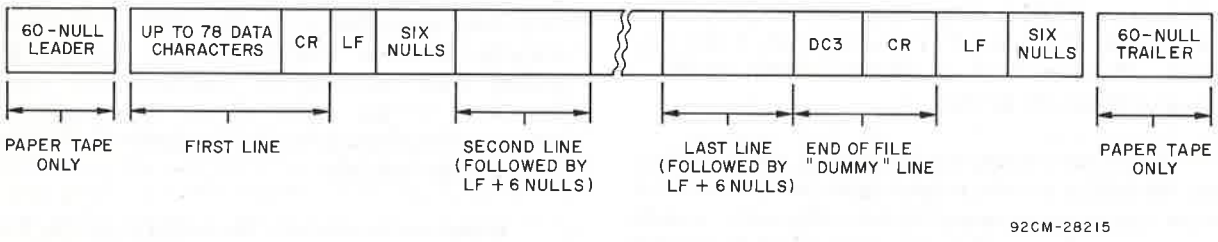


Fig. 27 — Tape file format.

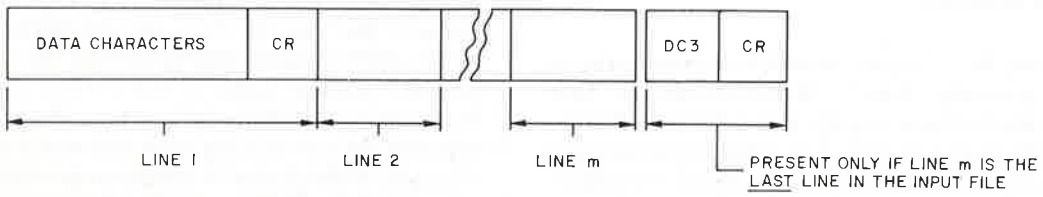


Fig. 28 — Memory buffer format.

When CRE is depositing keyboard data into its buffer, the ASCII code equivalent of each struck key (any printing character, and almost any control character - with exceptions as noted below) goes into memory and is also "echoed" back to the printer. However, CRE specifically ignores the LF key. Further, when the RETURN key is hit, the CR character goes into memory and a CR,LF pair of characters is echoed back to the printer to start a new line. Thus, the user terminates a line of keyboard input with a single carriage RETURN. Normally, then, the LF character should not appear at any point in the buffer.

Whenever CRE transmits a CR character to the terminal, it automatically appends to it the LF, six NULL field to maintain the tape format just discussed.

It is conceivable that due to a user error, one or more lines on the input file or in the buffer may exceed the 78 data character length restriction. For example, the input tape may have been erroneously prepared off line with the terminal operated in the LOCAL mode. Alternatively, data alterations in the buffer may have resulted in deleted CR's. (Note that each CR deleted in the buffer causes the concatenation of its adjacent lines.) CRE has the following provisions for handling lines which exceed the length restriction:

(1) Whenever CRE is outputting a line to the printer as the result of a user TYPE command, if the line exceeds 78 characters, a "LINE TOO LONG" message will also be printed.

(2) IF CRE encounters too long a line while writing from the buffer to the output tape, the line will be broken up, using as many 78-data-character records as are necessary - each terminated by an appended CR, LF, 6-NULL field.

(3) A line which is too long on the input file is truncated to 78 characters, with a CR appended, in the buffer.

Buffer Pointer

The total RAM space available for the buffer is generally partially filled. When CRE is first initialized, the buffer is empty. When data is added to the buffer (from the keyboard or from the input tape) the buffer expands. When data is deleted, the buffer contracts. CRE continually keeps track of the present extent of the buffer within the work space.

CRE maintains a virtual pointer which identifies some point between two characters in the buffer. This pointer has the same function as what is commonly called a "cursor". Most CRE operations are executed relative to this pointer. Further, several CRE operations exist specifically to alter the location of the pointer. It is essential, therefore, that the user be aware at all times of the current location of the pointer. Because the pointer is not visible, it is the user's responsiblity to keep track of where the pointer is. Often, its location is verified by asking CRE to type information in the buffer at the current pointer position. Alternatively, the user may first initialize the pointer to a known reference point (e.g., the beginning or end of a line, or the beginning or end of the buffer) and then move it relative to this known origin.

In illustrative examples, the location of the pointer is indicated with an arrow below and between the two buffer characters. For example, in

```
ABCDE
  ↑
```

the character before the pointer is B and that after the pointer is C.

Unless otherwise noted, whenever text is deleted from the buffer, the character sequence to be deleted exists either immediately to the right or immediately to the left of the pointer. After the deletion, the buffer has contracted by the number of characters deleted. If the field deleted is to the right of the pointer, the character immediately to the left of the pointer remains the same. The character to the right of the pointer then becomes the character that was immediately to the right of the deleted field. A corresponding statement can be made for deletion to the left of the pointer.

When text is inserted, the buffer expands. Unless otherwise noted, text is inserted between the two characters at the position of the pointer. After the insertion, the pointer is positioned immediately after the inserted text. Thus, the character to the right of the pointer remains the same.

The execution of many CRE operations starts at the present pointer position and proceeds either towards the end or towards the beginning of the buffer. CRE insures that the pointer cannot be moved past the present limits of the buffer. If the pointer reaches the beginning or the end of the buffer, the operation stops - leaving the pointer at that point. For example, if the pointer is positioned n characters from the end of the buffer and the user asks to move the pointer m characters to the right, with $m > n$, then the operation will stop after the buffer pointer has been incremented by only n.

CRE Command Operation

Command Strings

When control is first transferred to CRE, a "?" prompt is given, asking the user to identify the kind of I/O terminal in use. In response, the user types either P (for "punch") if a Teletype terminal is used, or any other character if a TI terminal is used. As will be explained further later, CRE must be aware of whether or not the output tape is to be punched because it has no direct means of turning the paper tape punch on and off. It must rely on the user to manually turn the punch on and off at appropriate time.

After the I/O terminal has been identified, CRE will print the initial message.

COSMAC TEXT EDITOR VERSION XX

and then follow this with its "→" user prompt character. If the I/O device is a Teletype terminal CRE will precede this → prompt by punching a leader of 60 NULL's on the output tape. (See later discussion on Punch Procedure.) The → prompt always indicates that CRE is ready to receive a new user command from the keyboard (having executed the previous one).

After receiving the →, the user types a sequence of one or more commands which CRE will execute in order. Most commands may be optionally delimited (ended) by a special termination control character. Commands which include text arguments of variable length must include this character to define the end of a text field. A CRE command string is always terminated by the occurrence of two successive ESCape or ALT MODE characters or its equivalent (the control character whose ASCII code is a hex 1B, if neither ESC or ALT MODE is labeled on the keyboard).

The I/O terminals discussed here, operate in the full duplex mode, in which the data path from the keyboard to the CDS is distinct from that between the CDS and the printer. Normally, a program merely "echoes" back to the printer what it has just received from the keyboard. However, **whenever CRE receives an ESC or ALT MODE character, it is echoed back to the printer followed by a $ - giving** a visual indication of the ESC key depression. Thus, a typical command string normally appears on the printer as

COMMAND1$COMMAND2$ . . . COMMANDn$$

where in most cases the separating ESC's are optional but the final pair is mandatory. (If ESC is not hit, the $ will not appear, of course.) **A command string must be terminated by two depressions of the ESC key.**

Command Formats

The heart of the command is a single letter mnemonic (such as "T" for TYPE, "I" for INSERT, etc.). In many cases, this letter may be optionally preceded by a decimal number (later denoted by n) indicating the number of characters or lines involved. Further, in some cases this number may be preceded by a minus sign (-) indicating a direction (from the present pointer position) toward the **beginning** of the buffer rather than toward the end (as is normally assumed). If no number is present, CRE assumes the value 1.

Given an arbitrary pointer location, the possible CRE interpretations for n are normally as follows:

(1) **Character Operations:** Positive n identifies the n characters to the right of the pointer (including control characters). Negative n identifies the n characters to its left. Unless otherwise noted, n=0 results in no operation.

(2) **Line Operations:** Positive n identifies all characters to the right of the pointer up to and including the nth CR encountered. If the pointer is in the middle of a line, the **first** line will constitute only the **remainder** of that line. Negative n identifies all characters to the left of the pointer up to but not including the -n+1st CR. If the pointer is in the middle of a line, the last line (in this set of lines) will consist of only those characters in the present line to the left of the pointer. Thus, n=0 specifically indicates the portion of the present line to the left of the pointer.

In certain cases a command mnemonic letter is followed by one or two variable-length text arguments (whenever the user needs to specify some sequence of characters to insert or to search for). All such arguments must be terminated by the ESC character (echoed as $). In subsequent discussion, an arbitrary text argument will be denoted by a symbolic statement such as "*text*".

Punch Procedure

Assuming the output file is to be punched on paper tape (i.e., the terminal device is a Teletype unit), CRE needs a mechanism to activate and deactivate the tape punch to properly bracket the information being output. Because CRE has no direct control of the punch, it must rely on the user. Whenever CRE wants to output to the punch, it types the message

START PUNCH, TYPE DEL

and then idles, awaiting the user's hitting of the DEL or RUBOUT key (whose hex code is 7F). The user should first manually turn the punch ON and then strike DEL. CRE will proceed to punch data. On completion, it will again idle awaiting another DEL which will indicate that the punch has been deactivated. The user should first manually turn the punch OFF and then strike DEL. CRE will then continue.

Note, in particular, that this process occurs at the initiation of an editing session when CRE automatically punches the initial 60-null leader on the output paper tape.

Correcting Command Typing Errors

A typing error in a command string may be corrected by use of the RUBOUT (DEL) character to "erase" previous characters already typed. Each time CRE receives a RUBOUT within a command string, it erases the last character from its stored version of the command string. Further, it echoes back to the terminal the character just erased. For example, suppose the user types the command string ABC$DE (each of the letters is a valid command mnemonic) followed by four rubouts. On the terminal, he would see

ABC$DEED$C

where the last four characters were those erased. The characters AB would then remain in CRE's stored command string register.

Clearly, any such erasures must occur before the double ESC character, which terminates the command string, is struck.

If CRE finds an invalid command while in execution of a command string (i.e., after the user has typed the double ESC), it returns to the user the error message

BAD COMMAND??"xxxx..xx$"

where xxx..xx reproduces that part of the command string which has not been executed.

Interrupting CRE Execution

The user may at any time stop CRE execution by depressing and holding the BREAK key on the keyboard. This key is used, for example, to stop a long typeout. On receipt of the BREAK, CRE stops execution at whatever point was reached and returns to the command input mode by issuing another prompt. To assure the clean entry of succeeding commands, the DEL key should be depressed to erase any erroneous noise characters that may have been entered as a result of the break.

After a BREAK, the user should normally verify or reinitialize the buffer pointer position before resuming further editing.

Filled Work Space Warning

If CRE determines that a command string threatens to use up the remaining work space, it will stop echoing keyboard input characters to the printer and will echo instead the the BELL control character -causing the I/O data terminal to ring its bell as a warning. The user should respond by erasing part of it with the RUBOUT key.

If CRE runs out of space during command execution, it will return the error message

MEMORY FULL "xxx...xx$"

where, again, xx...xx is a reproduction of the unprocessed part of the command string.

CRE Commands

This section contains a summary of the individual commands which CRE is designed to recognize. Each command is described with a specification of its acceptable format and an explanation of its execution. Examples are also given.

Single Commands

Pointer Control Commands

BEGINNING

Execution: Pointer repositioned to the beginning of the buffer.

END OF BUFFER

Format: Z
Execution: Pointer repositioned to the end of the buffer.

CHARACTER STEP

Format: nC
Execution: Step pointer right (or left)• by n characters.

LINE STEP

Format: nL
Execution: Step pointer down (or up) by n lines

Reading the Input Tape

APPEND

Format: A
Execution: Lines are read from the input file (continuing from the last line) and appended to the end of the buffer. The operation continues until one of the following occurs:

(1) End of file character detected (i.e., last line has been read).
(2) 3/4 of the remaining available space has been filled.
(3) 50 lines have been transferred.

The pointer is repositioned to the **beginning of the first appended line.** In large memory systems, multiple appends may be used to bring additional lines into the buffer.

Note: The keyboard BREAK key is ignored during execution of this command only.

Deletion Commands

DELETE

Format: nD
Execution: n characters right (or left) adjacent to the printer are deleted.

KILL

Format: nK
Execution: n lines right (or left) adjacent to the pointer are deleted.

Text Insertion and Data Manipulation

INSERT

Format: I*text*$
Execution: Typed *text* is inserted to left of present pointer position. The *text* may contain multiple lines.

SAVE

Format: nX
Execution: Copy n lines adjacent to the pointer into a special SAVE area **external** to the buffer. The pointer position is not changed. Previous contents of the SAVE area are overwritten. CRE types CAN'T SAVE if there is insufficient room in the SAVE area and it does not save any lines. CRE clears the SAVE area if n=0 (zero).

GET

Format: G
Execution: Equivalent to an INSERT, but using the present contents of the SAVE area as an implicit text argument. Note: SAVE and GET are especially useful in sequence as a copying mechanism - to MOVE text.

CRE dynamically allocates the available RAM work space to its SAVE area, stack area, and the buffer or editing area. Once lines have been SAVE'd, they remain in the SAVE area indefinitely until the next SAVE command overwrites them. If many characters have been SAVE'd, the area available for the buffer will be proportionally reduced. The SAVE area is not automatically cleared by a GET command. Several GET commands may be issued against the same SAVE area. It is good practice, therefore, to clear the SAVE area when it is no longer needed in order to make that area available to the buffer. This step is accomplished by typing 0X (zero-X).

If an attempt is made to save more lines than there is room for, CRE will type

CAN'T SAVE

and will not transfer any lines to the SAVE area.

• A positive (unsigned) n indicates the direction of right or down; a negative n indicates left or up for all commands.

FIND

Format: F*text*$
Execution: A search for the specified character sequence '*text*' occurs from the current pointer position toward the end of the buffer. It stops either when a match is first encountered or when the end of the buffer is reached. In the first case, the pointer ends positioned immediately after the matching string. In the latter case, a "CAN'T FIND" message is printed, and the pointer position is unchanged.

SUBSTITUTE

Format: S *search text* $ *substitute text* $
Execution: Operates as FIND does above (using *search text* as the search argument). However, on a match, the *substitute text* replaces the matching sequence - with the pointer positioned after the inserted text. The *substitute text* must not be omitted from the command.

Output Commands

TYPE

Format: nT
Execution: Type the n lines adjacent to the current pointer. The pointer position remains unchanged.

PUNCH

Format: nP
Execution: The n lines adjacent to the current pointer are written to the output tape and printed on the printer. The pointer position remains unchanged. The lines are not deleted from the buffer.

WRITE and DELETE

Format: nW
Execution: n is treated as positive. The n lines at the beginning of the buffer are written to the output tape, printed on the printer, and also deleted from the buffer. The pointer ends up positioned at the beginning of the remaining buffer.

END

Format: E
Execution: The entire buffer is written to the output tape and also printed on the printer. Any lines remaining on the input tape are then copied to the output tape and printed on the printer. Finally, if a teletypewriter is used, the 60 null trailer is punched out. CRE then reinitializes for a new editing session with buffer cleared and with the pointer positioned at the beginning of the work space.

NULLS

Format: N
Execution: If a teletypewriter is used, 60 nulls are punched. Otherwise, this command is ignored. The pointer is not changed.

Summary of CRE Commands and Control Characters

A summary listing of the foregoing commands together with the meaning of each one is given in Table X. A summary of the special CRE control characters is given in Table XI. The CRE error messages are summarized in Table XII.

Composite Commands and Nesting

CRE also permits the user to specify composite commands. A composite command is a command string (one or more commands) enclosed within angle brackets (<...>). A command string may be preceded by a decimal number indicating the number of times that the string within the brackets should be executed.

One composite command may include another. Thus, CRE permits the "nesting" of commands. For example,

B5<3C4<DI $>L>$$

causes replacement of the 4th through the 7th characters in the first 5 lines in the buffer by spaces. The pointer ends positioned at the beginning of the sixth line.

With nested commands, the user must be aware of the order in which commands will be executed and the number of times individual operations will occur. The following example should indicate the general algorithm. Other examples will be given later. Consider the command string

$$a<b<CS_1>c<d<e<CS_2>CS_3>CS_4>>$$

where the lower case letters represent numbers and where each CS_i, represents an elementary command string. Fig. 29 indicates CRE's flow chart for the execution of this command string. It is derived by properly pairing the angle brackets in the string. Notice, for example, that CS_2 is executed a number of times equal to the product of a, c, d, and e.

TABLE X – CRE COMMAND SUMMARY

Format	Meaning
B	Move pointer to BEGINNING of buffer.
Z	Move pointer to END of buffer.
nC	Step pointer right (or left) by n CHARACTERS.
nL	Step pointer down (or up) by n LINES.
A	APPEND lines to end of buffer from input tape. Reposition pointer to beginning of APPENDed area.
nD	DELETE n characters after (or before) pointer.
nK	KILL n lines after (or before) pointer.
I*text*$	INSERT *text* at present pointer position. (Position pointer after it.)
nX	Save n lines after (or before) pointer. (Pointer position unchanged.) Clears the SAVE area if n = 0.
G	GET the last SAVEd lines and INSERT them.
F*text*$	FIND the first occurrence of *text*, searching from present pointer position toward end of buffer. If found, position pointer after the match. If not, type CAN'T FIND.
S *search text* $ *substitute text* $	FIND *search text* and SUBSTITUTE *substitute text* for it.
nT	TYPE n lines after (or before) pointer. (No change in pointer location.)
nP	Output n lines after (or before) pointer. (Buffer and pointer remain unchanged.)
nW	WRITE (and delete from buffer) the first n buffer lines on the output type. (Pointer ends up at beginning of remaining buffer.)
E	END the editing session. Equivalent to an nW, with n equal to or greater than the number of buffer lines, followed by a copy of remaining input tape to output tape. If paper tape, terminate with NULL trailer.
N	If paper tape, punch 60 NULLs on output tape.

TABLE XI – SUMMARY OF SPECIAL CRE CONTROL CHARACTERS

(1) ESCAPE or ALT. MODE	Echoed as $. Optional command separator. Required after a TEXT field. Two required at the end of a command string.
(2) LINE FEED	Ignored on input. Inserted after CR on output.
(3) CARRIAGE RETURN	Line terminator character. Stored in buffer.
(4) NULL	Ignored on input. Set of six inserted after LF on tape output.
(5) RUBOUT or DELETE	Punch ON/OFF signal to CRE from user. Erases previous character in a command string. Ignored on tape input.
(6) DC3	End-of-file character. Inserted by user at end of a created file or read in from an existing input file.
(7) HORIZ TAB	Echoed as 1 to 8 spaces when typed. Converted to 1 to 8 spaces on tape output. Can begin a command implying a previous INSERT.
(8) BREAK	Pressing BREAK will terminate a long operation. Next, press RUBOUT or DELETE to get a prompt →.

Note:
Within a command string, but not within a text field, CRE ignores any inserted spaces or CR's. Spaces or CR's may be used to improve the readability of the command string, if desired.

TABLE XII – CRE ERROR MESSAGES

Message	Meaning
LINE TOO LONG	A line that CRE is attempting to TYPE has more than 78 characters.
BAD COMMAND?? "XXX . . . X$"	CRE has found an invalid command in a command string. XXX . . . X is that part of the string not executed.
<BELL>	Filled work space warning.
MEMORY FULL "XXX . . . X$"	CRE ran out of work space during an execution. XXX . . . X is the unprocessed part of the command string.
CAN'T SAVE	There is not enough room in the SAVE area.
CAN'T FIND	The specified character sequence was not found between the pointer's previous position and the end of the buffer.
ITERATION STACK FAULT	CRE ran out of stack space during execution of a command string. May indicate improperly paired brackets in the string.

To execute a nested command, CRE maintains a stack in part of the available work space. The amount of stack space required depends on the depth of nesting in the command, i.e., on the number of loops within loops, as in Fig. 29, which in turn depends on the depth of bracket-pairs-within-bracket-pairs in the command string. If CRE runs out of stack space during execution, it will issue the error message:

ITERATION STACK FAULT.

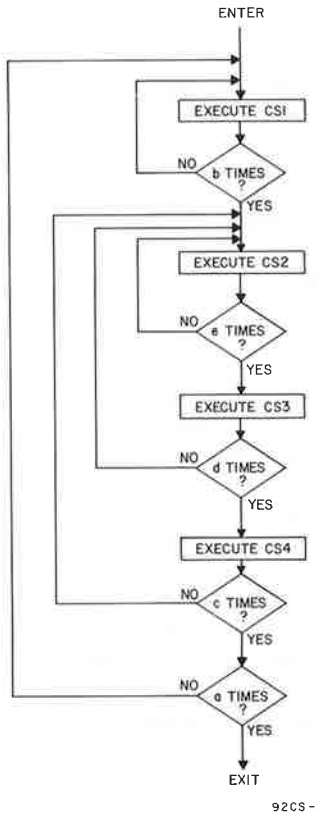


Fig. 29 – Execution of nested composite command.

This error message is most likely to occur if the brackets in the command string are not paired properly. In particular, it occurs if a bracket is missing.

Note that if the user fails to terminate a *text* string with the required ESC character, all subsequent characters until an ESC does occur will be treated as part of the presumed *text* string. Thus, it is quite possible that a missing ESC in a nested command string could also result in the "improperly-paired-brackets" error message, ITERATION STACK FAULT.

Horizontal Tabs

CRE assumes an implicit horizontal tab stop after every eight character positions in a line. If the user types a HORIZ TAB character (CTRL and I) as part of a TEXT field, CRE will insert this character into its buffer, but it will echo back to the printer a sufficient number of spaces to reach the next implied tab stop. HORIZ TAB characters read from the input file are loaded into the buffer as is. On output, each HORIZ TAB buffer character is converted into the required number of spaces, extending the line length in the process. Thus, HORIZ TAB characters cannot appear on the output tape. The TAB character can be used to produce straight columns in a source file.

Note: As a special case, CRE interprets a text beginning with a HORIZ TAB character as if an INSERT command had preceded it.

Additional Note

Normally, the INSERT of a non-existent text field (i.e., the command I$) results in no operation. Further, it is normally illegal to precede an INSERT command with a numeric argument. However, the

specific command nI$ (combining the two) is legal. It causes the insertion of a single character whose ASCII decimal value is n (modulo 128).

For example, 10I$ will cause insertion of a LF character (hex 0A).

Using CRE

In this section, information is given on the development and manipulation of a COSMAC file through use of the COSMAC Resident Editor (CRE). In addition, some useful common sequences are given to illustrate CRE's data manipulation facilities.

Loading and Operating CRE

The following steps are required to use CRE:

1. Load the program as follows:

   a. For a TI terminal in the LINE mode.
      1) Press RESET, then RUN U, then CR - get a *.
      2) Mount CRE program cassette and rewind it.
      3) Press LOAD/FF to advance to the first file.
      4) Press CONT/START to start reading the file.

   b. For a Teletype terminal in the LINE mode.
      1) Press RESET, then RUN U, then CR - get a *.
      2) Mount CRE tape in reader.
      3) Place the READER Control switch in the Manual position.
      4) Press START on the tape reader.

2. Mount input file (if any) in reader.

3. Mount a blank cassette for the output file. If paper tape, do not turn punch ON yet.

4. Type $U(CR) to start CRE.

5. CRE begins by typing "?" asking user to identify the type terminal in use. In response, type P for a Teletype terminal or any other character for a TI terminal.

6. CRE will then give the → prompt indicating that it is ready to accept commands.

File Development and Manipulation

Creating a File

A file is created by a repeated sequence of the following steps:

(1) File buffer from keyboard with sequence of INSERT's.
(2) WRITE buffer to output file.

A single I command may take as an argument a text string of abritrary length. Thus, many lines may be inserted with a single I command. Each line is terminated by pressing the RETURN key. A typical INSERT will thus appear on the printer as

```
→ I line 1      ..Insert command and 1st line of
                .. text.
line 2          ..Additional lines
.               ..Additional lines
.               ..Additional lines
.               ..Additional lines
line n          ..Last line of text
$$              ..End of insert; double
                ..ESCAPE
```

because each CR is echoed as CR, LF. Such commands may be sequenced until the buffer is nearly filled. These sequences are then normally followed by an nW (WRITE) command or BnP, with n ⩾ the number of lines in the buffer. By use of the W command, the buffer is cleared after the WRITE to the output file and is ready for a new set of INSERT's.

The last line inserted should be followed by the insertion of a terminating dummy line consisting of the single character DC3 (CRTL and S), followed by the usual RETURN to assure proper tape format, as discussed earlier.

If the output tape is a paper tape, a final N command will add the trailer of NULL's.

Adding to a File

A section is added to an existing file by first copying the portion before the insert, and finally copying the portion after the insert. The first copy involves one or more APPEND's followed by WRITE's, up to the APPEND which reads in the section of the input file which contains the insertion point. Note that appending to the end of a file may also be considered as an insertion just before the last DC3 terminating line.

Assuming the insert point is arbitrarily located within the buffer, several variations exist for adding text material. For any of these variations, the pointer must first be moved to the insert point. Then, a sequence of INSERT's is made at that point, particularly if the amount of the inserted material is small. Alternatively, one could SAVE all lines following the pointer (with an nX, n sufficiently large), delete them with an nK command, and then WRITE the data remaining in the buffer with an nW (n sufficiently large). The buffer then becomes empty with all records preceding the addition written to the output tape. Additional INSERT's and WRITE's may now be made. Finally, a GET followed by a WRITE will attach the material after the insert point. Now if there is more unread material on the input file, the GET may be followed directly by an END command. This command will automatically copy the remaining input tape.

In summary, one inserts material into an existing file by beginning with a copy sequence (a series of APPEND's followed by WRITE's). Then with the pointer positioned properly, one may execute nX nK nW (n sufficiently large). Now, one operates in the CREATE mode with INSERT's followed by WRITE's. Finally a GET or GnW will complete the sequence.

When appending to the end of a file, one has the alternative of removing, after the last APPEND, the dummy termination line via a Z-1K command string. Operation then is as in the CREATE mode. For this case, the final DC3 line must later be inserted to the end of the file.

Deleting a Section in a File

To delete a section in a file, first copy up to the deletion point, as previously discussed. Lines to be omitted may then be explicitly deleted from the buffer (by nK, with pointer properly positioned). If further lines to be deleted exist on the input file, further APPEND's are required.

Note that the nP command causes a WRITE from the present pointer position. Thus, text may be implicitly deleted by moving the pointer past it and issuing an nP command (n positive). After the remainder of the buffer has been written, the command string BnK (n sufficiently large) will erase the buffer.

Moving a Section in a File

Assume that the file section to be moved is sufficiently small. If the movement is toward the end of the file, the following sequence may be used:

(1) Copy input file up the section to be moved.
(2) SAVE the section to be moved. Then DELETE it in the buffer.
(3) Continue copying the input file up to the insertion point.
(4) GET and WRITE the SAVE'd section.
(5) Copy the remaining part of the input file.

If the movement is toward the beginning of the file, one must first find the section to be saved, SAVE it, DELETE it, and then **reinitialize** the input file. Then, the sequence of steps 3, 4, and 5 above will effect the insertion.

Note again that the P command may be used to WRITE from an arbitrary point in the buffer. Note also that the material thus written is not deleted from the buffer.

Several complications of this simple procedure can occur. First, the material to be moved may overlap two APPEND's. In this case, one does not SAVE until the second APPEND has been executed. Second, the material to be moved may consist of a substantial portion of the input file so large that it must first be copied on to a third temporary tape which might be called an "insertion file". If this condition exists, the user should be sufficiently familiar with CRE so that he will be able to create and use this special temporary file.

Modifying a Section in a File

By now the reader should be reasonably familiar with the commands APPEND, PUNCH, WRITE, END, INSERT, SAVE, GET, and NULL's.

The most common use of CRE is to modify the contents of a file at a given point (typically, to correct an error). To make such a modification, the user must first read that section of the file into the buffer. Normally, a copy of the initial portion of the file is necessary, up to the APPEND which brings into the buffer the section to be modified. Now, the remaining CRE commands are available to effect the modification. After the change is made, the process is terminated with an END command or a WRITE command as appropriate.

Some Command Examples

Below are several examples of useful command sequences to further acquaint the reader with CRE's data manipulation facilities. In each example a command string is given and followed by a short explanation of what it will do. For clarity all zeros are slashed (Ø) to distinguish them from the letter O in these examples.

(1) Assume the pointer is arbitrarily positioned within a line in the buffer:

ØLT Types the entire line leaving the pointer at its beginning.

ØTT Also types the entire line, but leaves the pointer unchanged.

ØK Erases the portion of the line to the left of the pointer.

K Erases the portion of the line to the right of the pointer.

ØLK Erases the entire line.

For each of the following command sequences, it is assumed that n is sufficiently large.

BnK Erases the entire buffer.

ØK Erases the entire SAVE area.

BnT Prints the entire buffer.

(2) Assuming the pointer is positioned at the beginning of a line in the buffer,

nXnKZ-mLG

will move the next n lines to m lines from the end of the buffer and erase them from their original position.

(3) The command Bn<mCI $L >, for n sufficiently large, inserts a field of spaces in all lines, at a point m characters from the beginning of each line.

(4) One can also scan the entire buffer with a FIND or SUBSTITUTE command by similarly using a sufficiently large numeric argument (called n below). The command will terminate when the end of the buffer is found with a CAN'T FIND message. For example:

Bn<S*field1*$*field2*$>

will replace **all occurrences** of *field1 by field2*.

Bn<F*text*$- mD>

will delete **all occurrences** of *text*, if m=the length of the *text* field.

Bn<F*text*$ØLT>

will print all lines containing *text*.

Bn<F*text*$ØLK>

will delete all lines containing *text*.

Bn<F;ICR>

will break all lines containing semicolons into as many lines as there are semicolons - each terminating in a semicolon. (Note: In this case, any line originally ending in a semicolon will be followed by a "line" containing zero characters).

Bn<S TABL>

will replace the first space in every line in the buffer by a horizontal tab control character.

Bn<A5ØT5ØK>

will perform the following n times: append in the next (first) section type it, and delete it from the buffer. This command string can be used to type a long file that can't be held all at once in the buffer. It is particularly useful in typing the listing output file on the assembler.

Appendix A -
CDS II CDP18S005 Backplane Wiring Schedule

Location	Pin No.	Memory (1-9)	Address Latch and Bank Select (10)	CPU (12)	I/O Decode (13)	I/O [3] (14-24)	Control (25)	Pin No.
	A	–	BSE–P	TPA–P	TPA–P	TPA–P	TPA–P	A
	B	SPARE	–	TPB–P	TPB–P	TPB–P	TPB–P	B
	C	DB0–P	BS9–P	DB0–P	DB0–P	DB0–P	DB0–P	C
	D	DB1–P	RNU–P	DB1–P	DB1–P	DB1–P	DB1–P	D
	E	DB2–P	BS8–P	DB2–P	DB2–P	DB2–P	DB2–P	E
	F	DB3–P	BS7–P	DB3–P	DB3–P	DB3–P	DB3–P	F
	H	DB4–P	BS6–P	DB4–P	DB4–P	DB4–P	DB4–P	H
	J	DB5–P	BS5–P	DB5–P	DB5–P	DB5–P	DB5–P	J
	K	DB6–P	BS4–P	DB6–P	DB6–P	DB6–P	DB6–P	K
	L	DB7–P	BS3–P	DB7–P	DB7–P	DB7–P	DB7–P	L
	M	A0–P	BSD–P	A0–P	SEL0–P	–	A0–P	M
	N	A1–P	BSF–P	A1–P	SEL1–P	–	A1–P	N
	P	A2–P	A15–P	A2–P	SEL2–P	N=4–P	A2–P	P
	R	A3–P	A14–P	A3–P	SEL3–P	N=5–P [4]	A3–P	R
	S	A4–P	A12–P	A4–P	SEL4–P	N=6–P	A4–P	S
	T	A5–P	–	A5–P	SEL5–P	–	A5–P	T
	U	A6–P	–	A6–P	SEL6–P	–	A6–P	U
	V	A7–P	–	A7–P	SEL7–P	–	A7–P	V
	W	MWR–N	BS0–P	MWR–N	N=7–P	N=7–P [5]	–	W
	X	BSN–P [1]	MBDS–N	CPU PWR	N=6–P	EF4–N	RUN–N	X
	Y	V_{DD}	V_{DD}	V_{DD}	V_{DD}	V_{DD}	V_{DD}	Y
	Z	GND	GND	GND	GND	GND	GND	Z
	1	TPA–P	TPA–P	DMAI–N	N=1–P	DMAI–N	–	1
	2	TPB–P	UA15–N	DMAO–N	N=2–P	DMAO–N	DMAO–N	2
	3	SPARE	BSC–P	ANY I/O–P	N=3–P	–	ANY I/O–P	3
	4	–	BSB–P	INT–N	N=4–P	INT–N	RNU–P	4
	5	MRD–N	BSA–P	MRD–N	MRD–N	MRD–N	MRD–N	5
	6	A12–P	A12–P	Q–P	N=5–P	Q–P	Q–P	6
	7	A11–P	A11–P	SC0–P	–	SC0–P	SC0–P	7
	8	A10–P	A10–P	SC1–P	–	SC1–P	SC1–P	8
	9	A9–P	A9–P	CLEAR–N	TLIO–N	–	CLEAR–N	9
	10	A8–P	A8–P	WAIT–N	–	–	WAIT–N	10
	11	–5 V	A0–P	–	–	–5 V	EX CLR–P	11
	12	EX WAIT	A1–P	EX CLK	–	–	EX WAIT–P	12
	13	CLK OUT	A2–P	CLK OUT	RESET–OP	RESET–OP	RESET–OP	13
	14	–	A3–P	N0–P	N0–P	–	N0–P	14
	15	–	A4–P	N1–P	N1–P	–	N1–P	15
	16	RESET–OP	A5–P	N2–P	N2–P	–	N2–P	16
	17	–	A6–P	EF1–N	–	EF1–N	–	17
	18	–	A7–P	EF2–N	–	EF2–N	–	18
	19	V_{DD} [2]	BS2–P	EF3–N	–	EF3–N	EF3–N	19
	20	+12 V	BS1–P	EF4–N	+12 V	+12 V	+12 V	20
	21	V_{DD}	V_{DD}	V_{DD}	V_{DD}	V_{DD}	V_{DD}	21
	22	GND	GND	GND	GND	GND	GND	22

Notes

[1] BSN–P: No printed wires. Wire-wrap to user's choice, BSO through BSF.
[2] Location 8 only.
[3] Locations 19 and 20: all pins open except V_{DD} and GND.
[4] Location 24 only (Disk interface).
[5] Location 14 only (Terminal interface).

Wire-Wrap Connections

RAM SELECT	BS0–P	10–W to 8–X
ROM SELECT	BS8–P	10–E to 9–X
TERM. SELECT	SEL0–P	13–M to 14–M
DISK SELECT	SEL0–P	13–M to 24–M
2-LEVEL I/O	TLIO–N	13–9 to 13–22 (Jumper)
I/O DECODE	RNU–P	13–7 to 24–4

Appendix B - Instructions for Converting a Model 33 Teletype Terminal from Half-to-Full-Duplex Operation and from 60-mA to 20-mA Operation

For a Teletype terminal connected for half-duplex operation, the following modifications can be made to convert it to full-duplex operation.

1. Locate the black terminal strip in the back. See Fig. B1.
2. Move the brown/yellow and white/blue wires from pins 3 or 4 to pin 5.

For Teletype terminals, connected for 60-mA operation, the following modifications can be made for 20-mA operation.

1. Move the violet wire from pin 8 to pin 9.
2. Move the blue wire connected to the current source resistor (a flat green resistor with four tabs located to the right of the keyboard) from the 750-ohm tab to the 1450-ohm tab.

Fig. B2 gives the detailed interface circuitry between the CDS logic signals and the pin connections for the Teletype terminal in the full-duplex mode. Note particularly the isolation of the two Teletype (TTY) current loops. Also shown in Fig. B-2 is the detailed interface circuitry between the CDS logic signals and the pin connections for an EIA RS232C type data terminal.

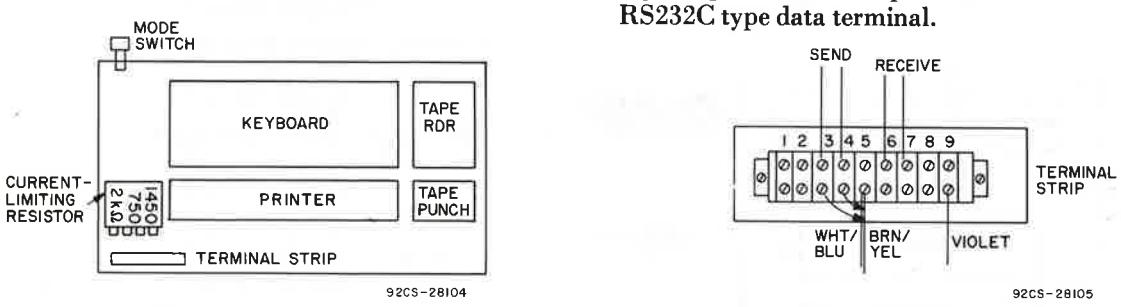


Fig. B1 — Location of and connections to terminal strip for Model 33 Teletype data terminal showing connections for 20-mA full-duplex operation.

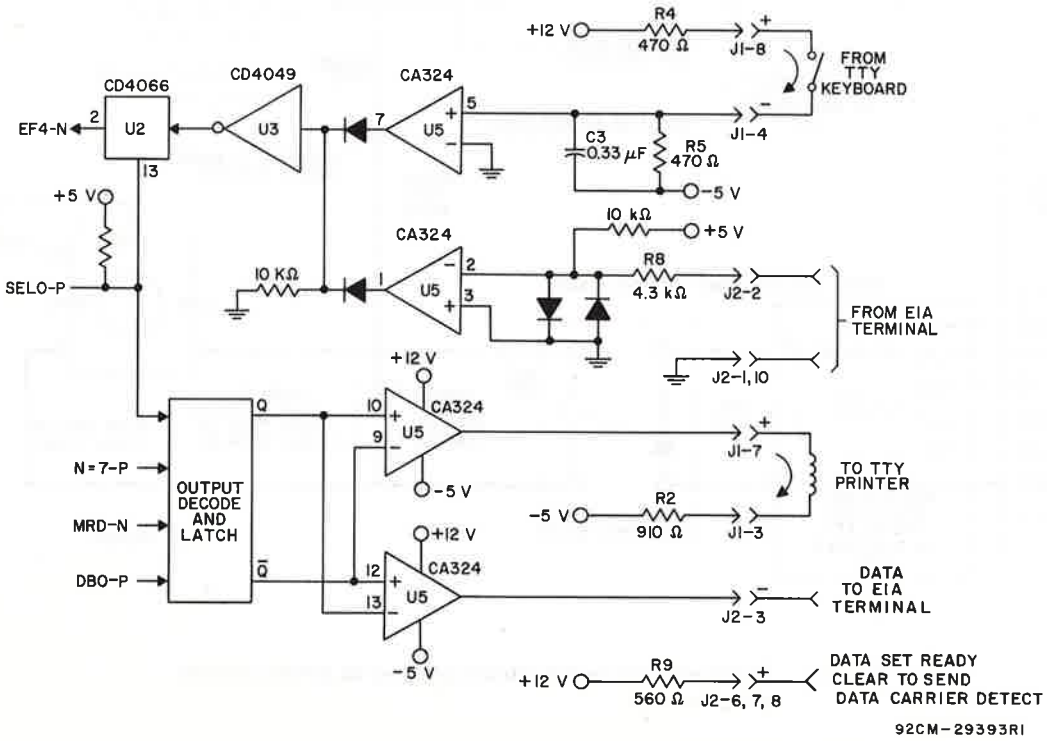


Fig. B2 — Detail of CPU-Terminal Interface (See Appendix D).

Appendix C - Adding Teletype Remote Reader Control

A simple wiring change inside the conventional Model 33 Teletype terminal permits the paper-tape reader to be operated under control of an external COSMAC (program-derived) signal. The modifications are indicated in Fig. C1. Two additional components must be appropriately mounted: an electronic relay and a switch. Wiring to be added is indicated by the bold line in the diagram. Note that the wiring connects the added switch and relay to points on the front mode switch and in the array of white plastic Molex connectors located in the back of the unit under the cover. Note also that one brown wire must be broken and reconnected as shown.

The Terminal Interface module contains the logic necessary to permit a COSMAC program to control the paper-tape reader. With the added reader control switch in the remote or open position, a program may turn the reader on and off. In the manual or closed position, the reader can be controlled only manually, by means of the original reader control switch on the tape reader. Note particularly that this latter switch must also be activated (in the start position) in order for the remote program control to operate properly.

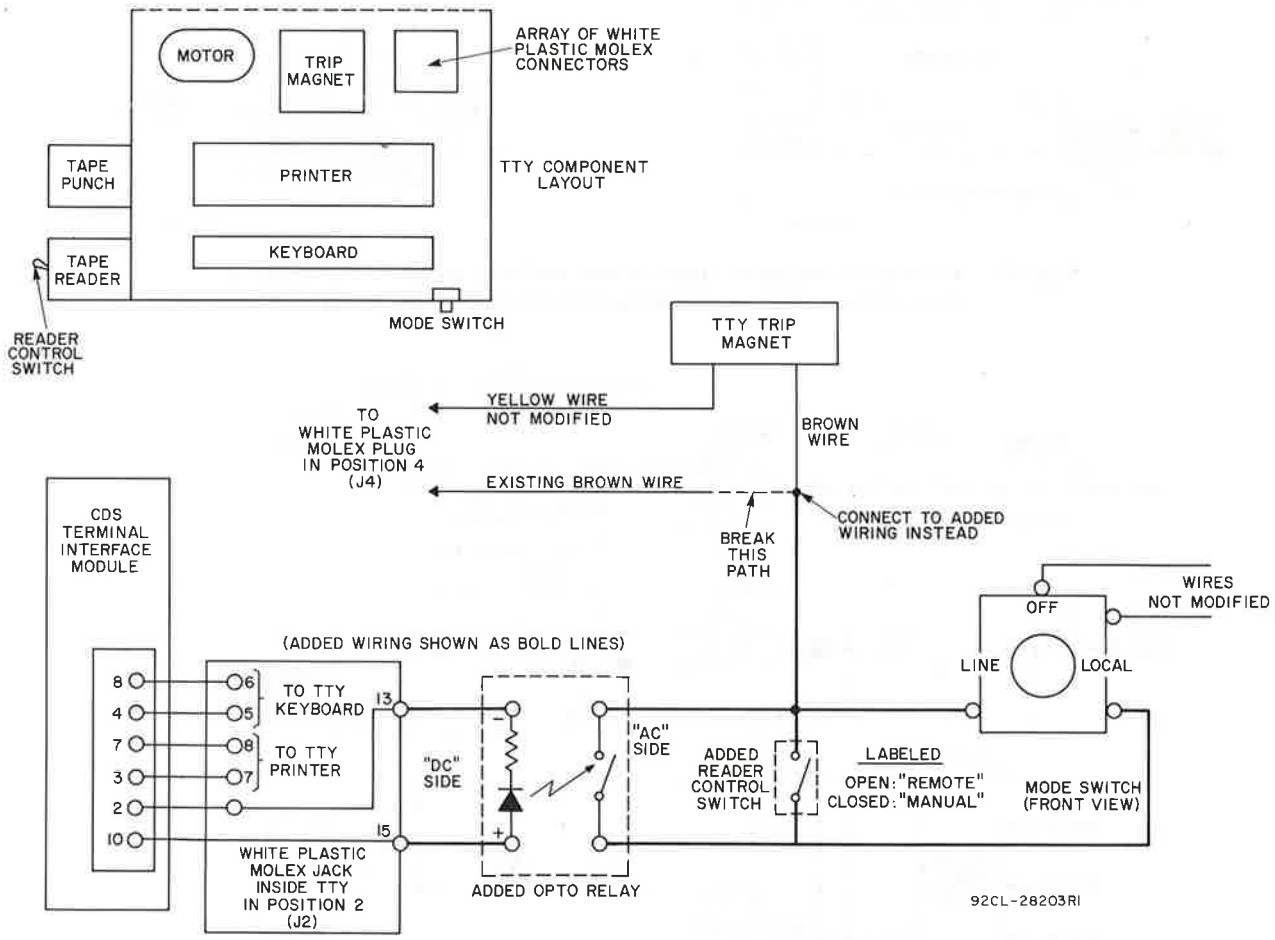
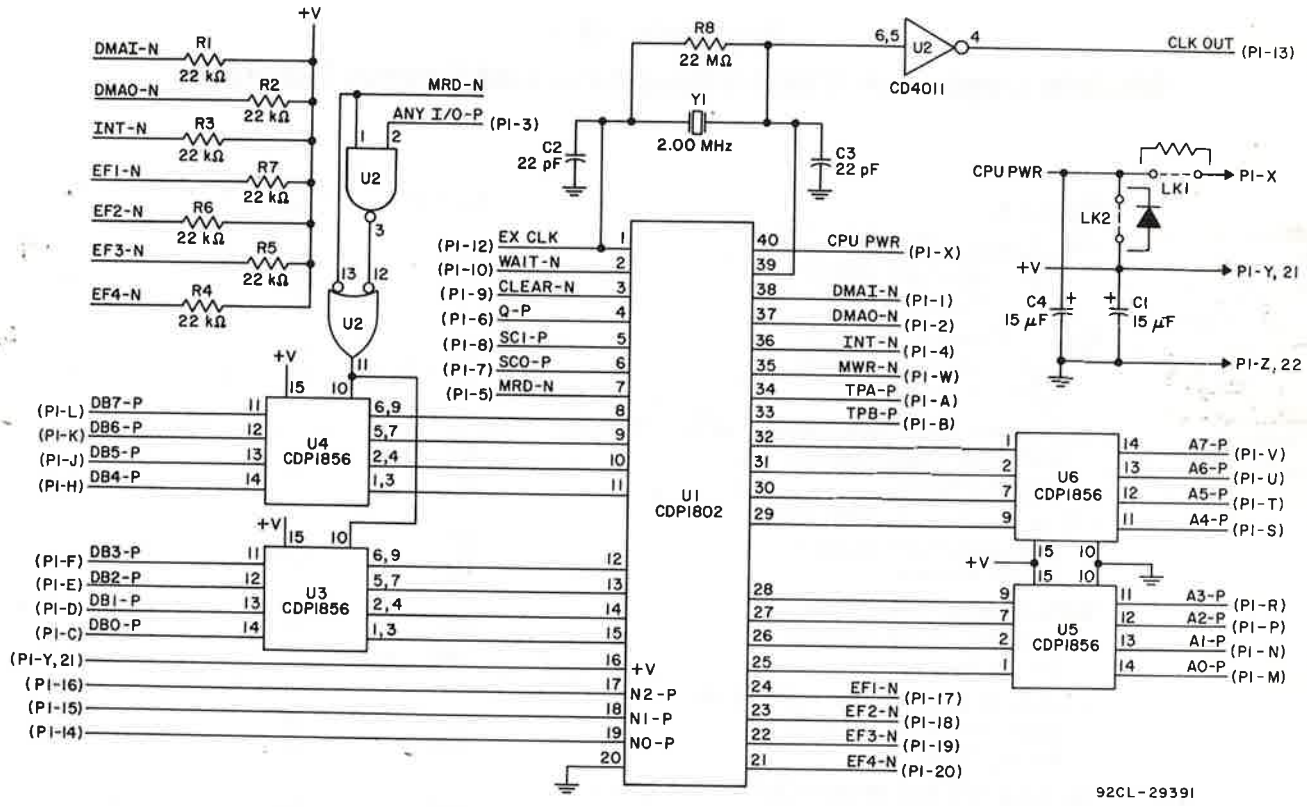


Fig. C1 — Teletypewriter modifications required to permit remote reader control.

Appendix D -
Module Logic and Circuit Diagrams and Layout Diagrams



#12

Fig. D1 — CPU module CDP18S102 logic and circuit diagram.

Parts List for Fig. D1 (CDP18S102)

C1, C4 = 15 μF, ± 20%, 20 volts
C2, C3 = 22 pF, ± 20%, 20 volts
R1 through R7 = 22 kilohms, ± 5%, 1/4 watt
R8 = 22 megohms, ± 5%, 1/4 watt
U1 = CDP1802CD
U2 = CD4011BE
U3, U4, U5, U6 = CDP1856D
Y1 = crystal, 2.00 MHz

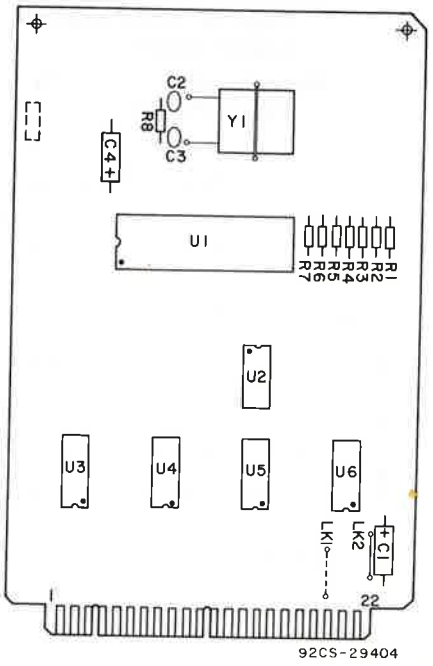


Fig. D2 — CPU module CDP18S102 layout diagram.

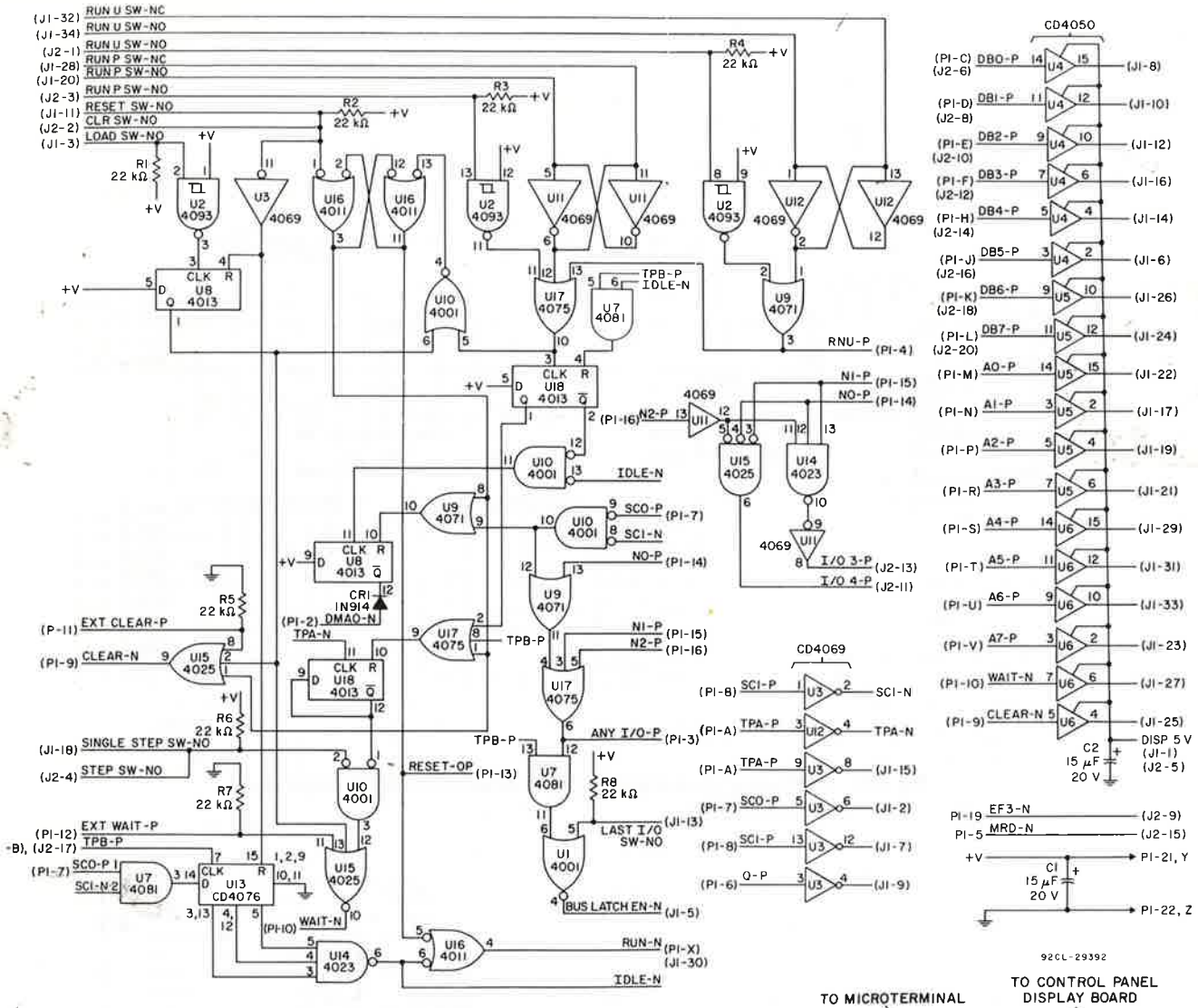


Fig. D3 — Control module CDP18S103 logic and circuit diagram.

Parts List for Fig. D3 (CDP18S103)

C1 = 15 μF, ± 20%, 20 volts
CR1 = 1N914
J1 = connector, 34 pin
J2 = connector, 20 pin
R1 through R8 = 22 kilohm, ± 5%, 1/4 watt
U1, U10 = CD4001BE
U2 = CD4093BE
U3, U11, U12 = CD4069BE
U4, U5, U6 = CD4050BE
U7 = CD4081BE
U8, U18 = CD4013BE
U9 = CD4071BE
U13 = CD4076BE
U14 = CD4023BE
U15 = CD4025BE
U16 = CD4011BE
U17 = CD4075BE

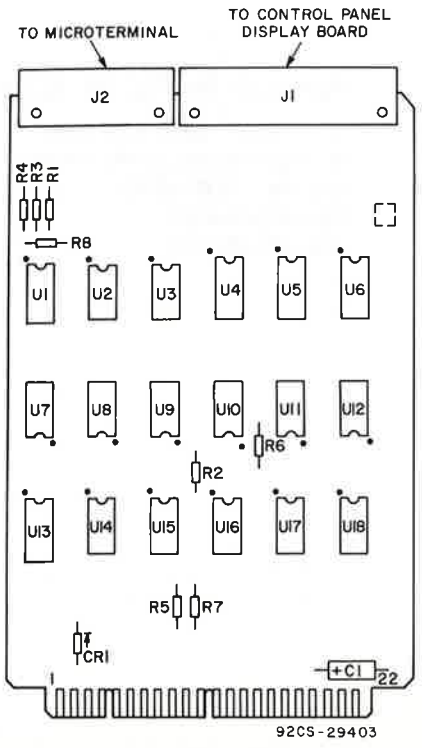


Fig. D4 — Control module CDP18S103 layout diagram.

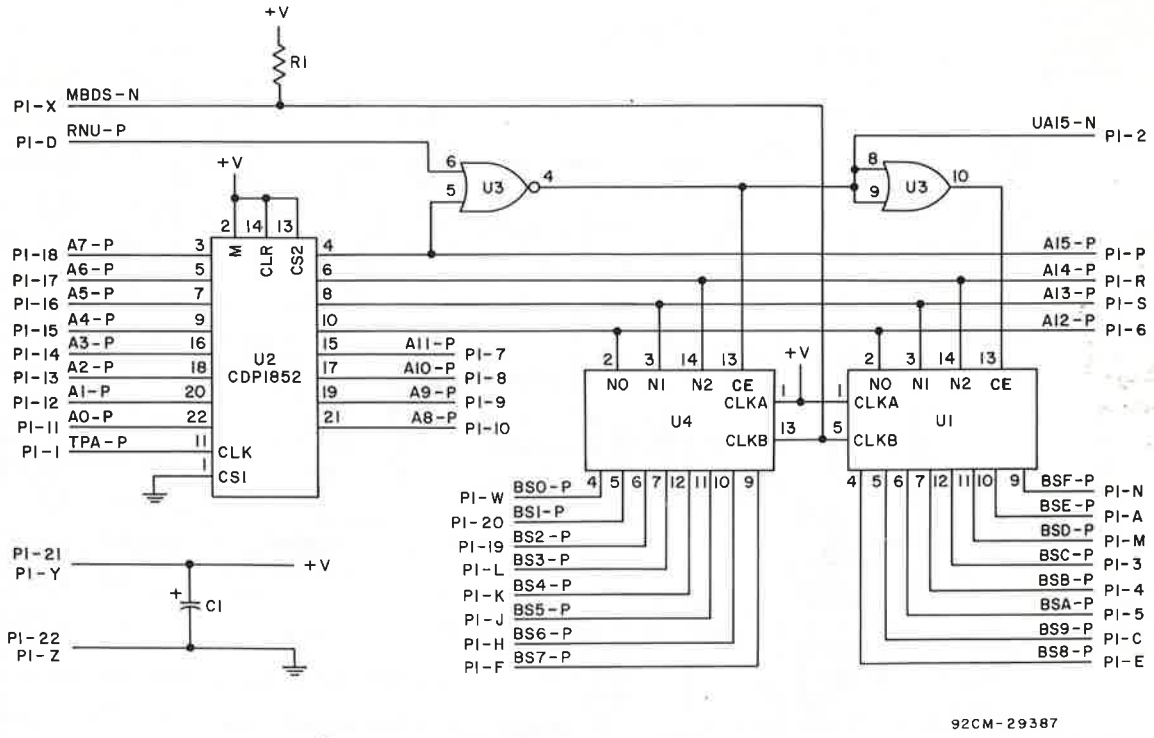


Fig. D5 — Address latch and bank select module CDP18S206 logic and circuit diagram.

Parts List for Fig. D5 (CDP18S206)

C1 = 15 μF, 20%, ± 20 volts
R1 = 22 kilohm, ± 5%, 1/4 watt
U1, U4 = CDP1853D
U2 = CDP1852D
U3 = CD4001BE

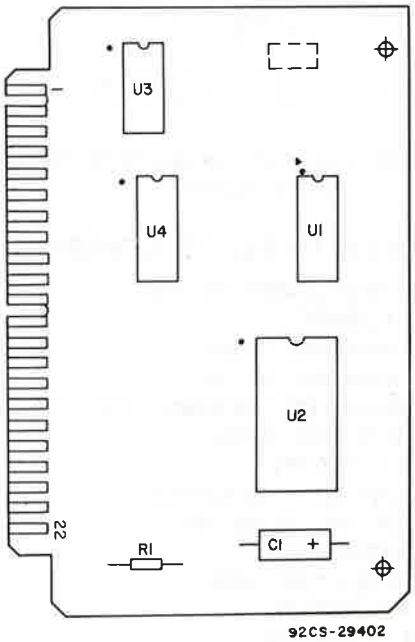


Fig. D6 — Address latch and bank select module CDP18S206 layout diagram.

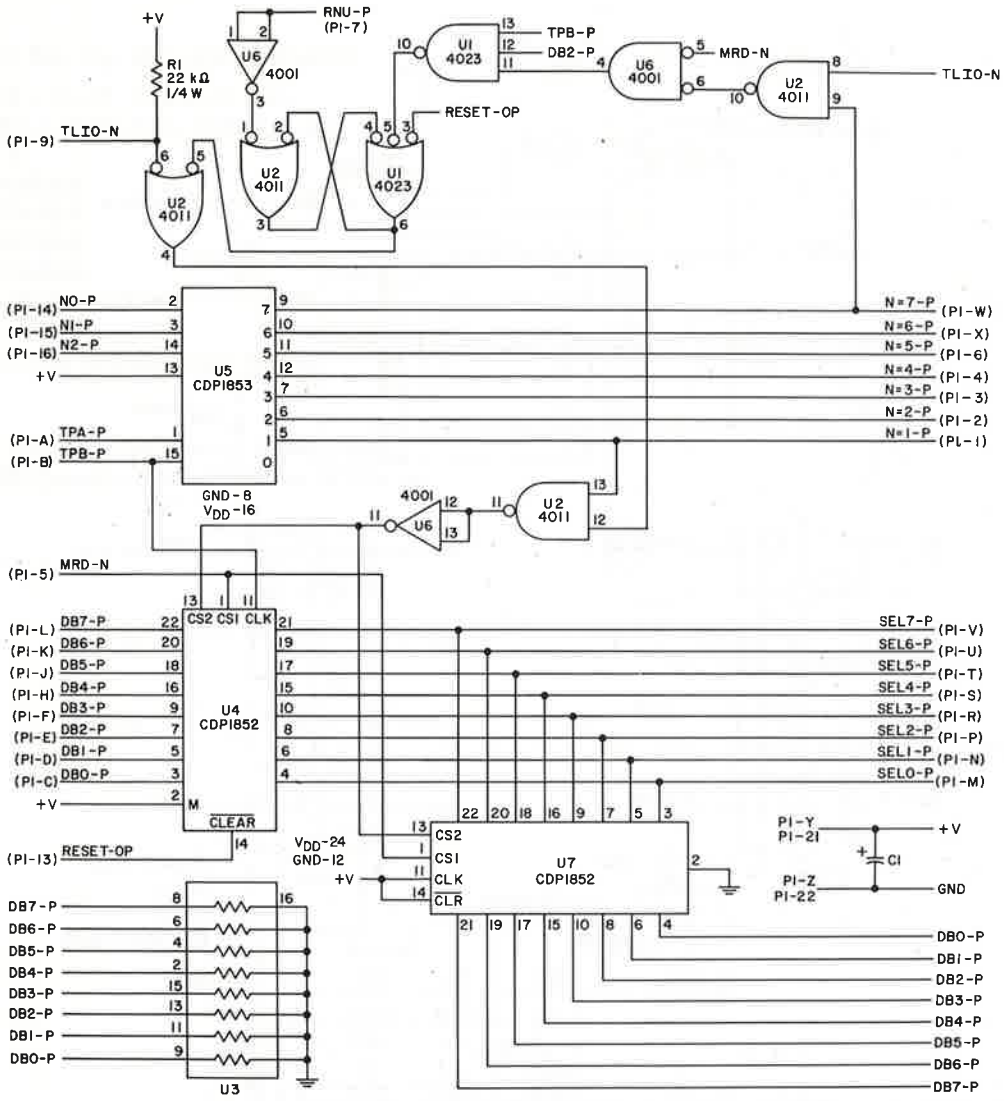


Fig. D7 — I/O decode module CDP18S509 logic and circuit diagram.

Parts List for Fig. D7 (CDP18S509)

C1 = 15 μF, ± 20%, 20 volts
R1 = 22 kilohms, ± 5%, 1/4 watt
U1 = CD4023BE
U2 = CD4011BE
U3 = resistor module, 15 kilohms
U4, U7 = CDP1852D
U5 = CDP1853D
U6 = CD4001BE

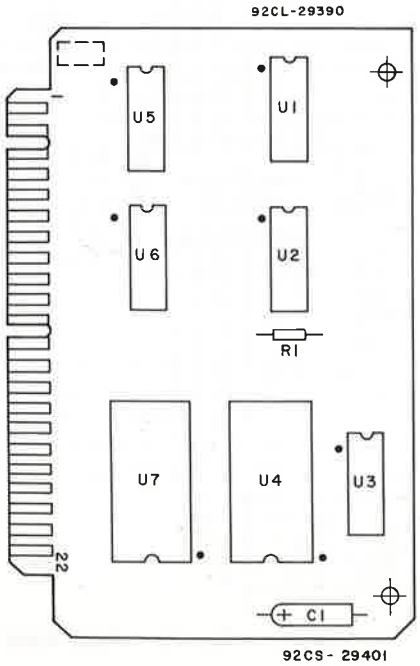


Fig. D8 — I/O decode module CDP18S509 layout diagram.

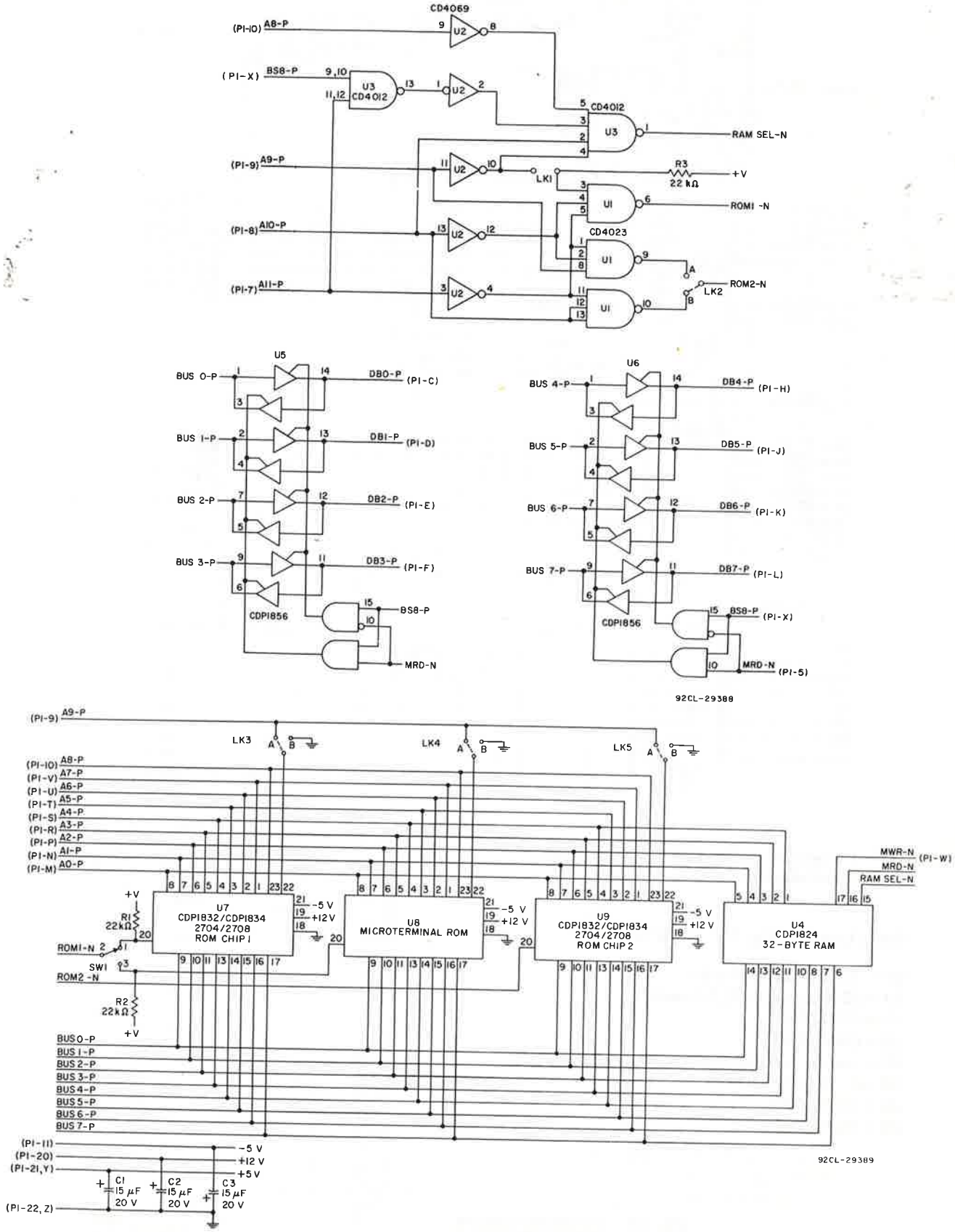
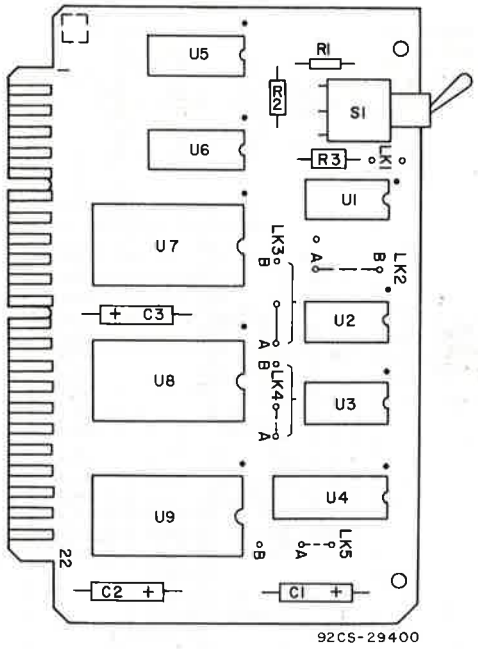


Fig. D9 – ROM/RAM module CDP18S401 logic and circuit diagram.

Parts List for Fig. D9 (CDP18S401)

C1, C2, C3 = 15 μF, ± 20%, 20 volts
R1, R2, R3 = 22 kilohms, ± 5%, 1/4 watt
S1 = SPDT
U1 = CD4023BE
U2 = CD4069BE
U3 = CD4012BE
U4 = CDP1824D
U5, U6 = CDP1856D
U7, U9 = 2708
U8 = Socket for Microterminal ROM

Note:
S1 UP enables Microterminal ROM in U8.
S1 DOWN enables UT20 ROM's in U7 and U9.



#9

Fig. D10 – ROM/RAM module CDP18S401 layout diagram.

Link Connections:

ROM			LINKS	
U7	U8	U9	LK1	LK2
512	512	512	short	A
♦ 1 k	512	1 k	open	B
1 k	1 k	1 k	open	B
♦ 1 k	512	512	open	A
1 k	1 k	512	open	A
■ 512	512	1 k	short	B

ROM	U7 LK3	U8 LK4	U9 LK5
2704	B	B	B
2708	A	A	A
CDP1832	A or B	A or B	A or B
CDP1834	A	A	A

♦ The low-order U8 addresses are not contiguous with the U9 addresses and the second 512 addresses overlay the first 512 (wrap).

■ The low-order 512-byte addresses are not contiguous with the U9 addresses and do not wrap.

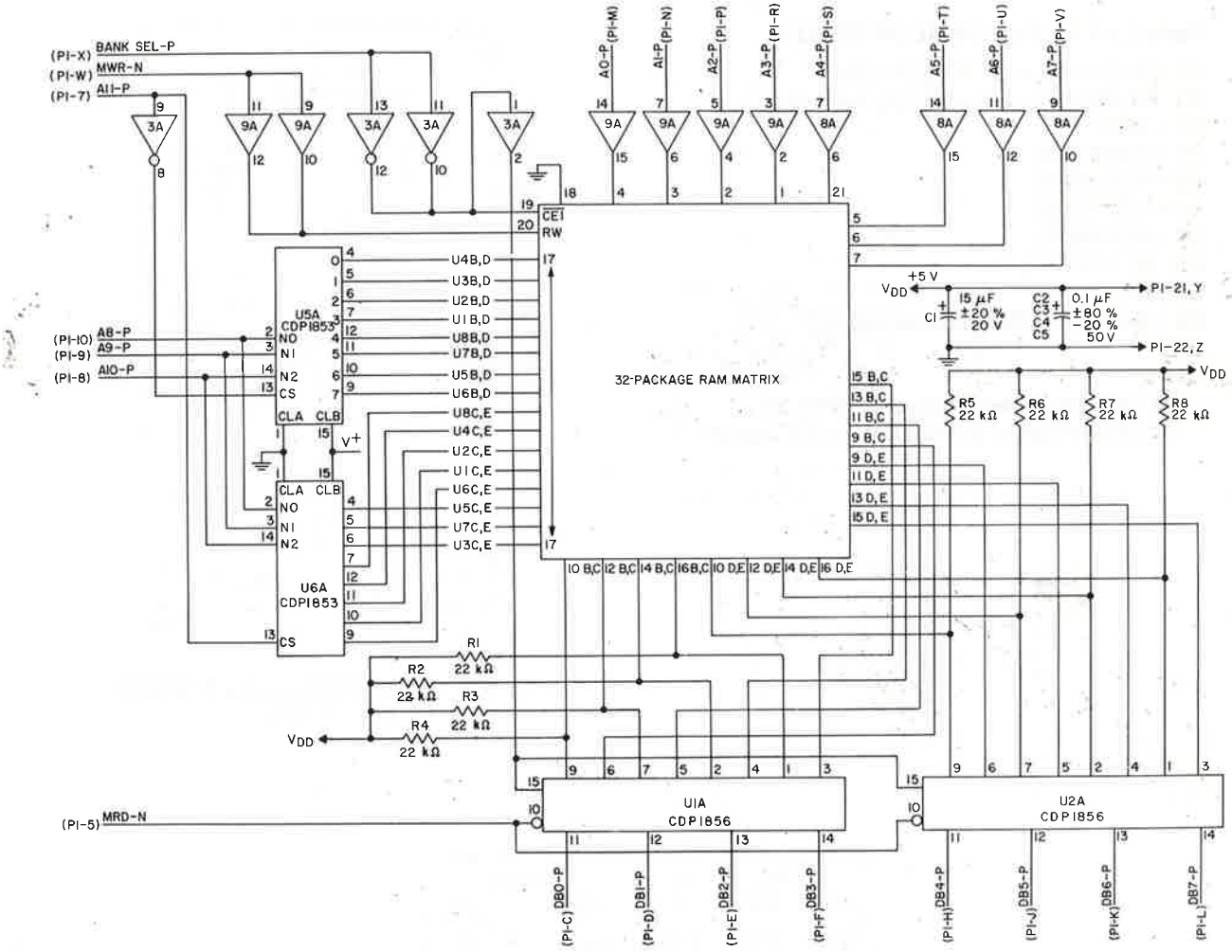


Fig. D11 -- 4-kilobyte RAM module CDP18S205 logic and circuit diagram.

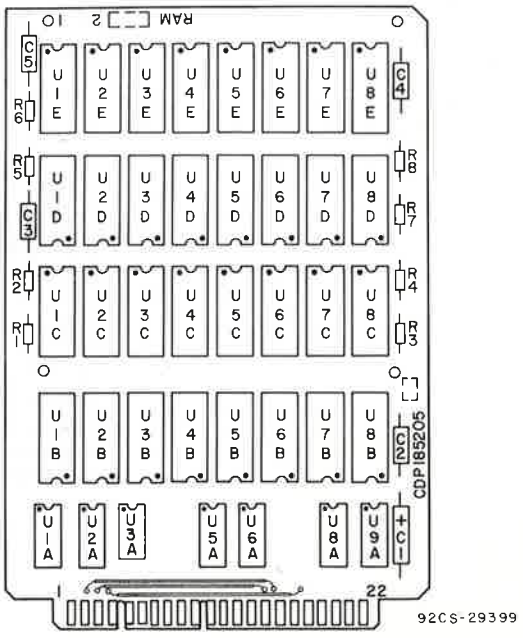
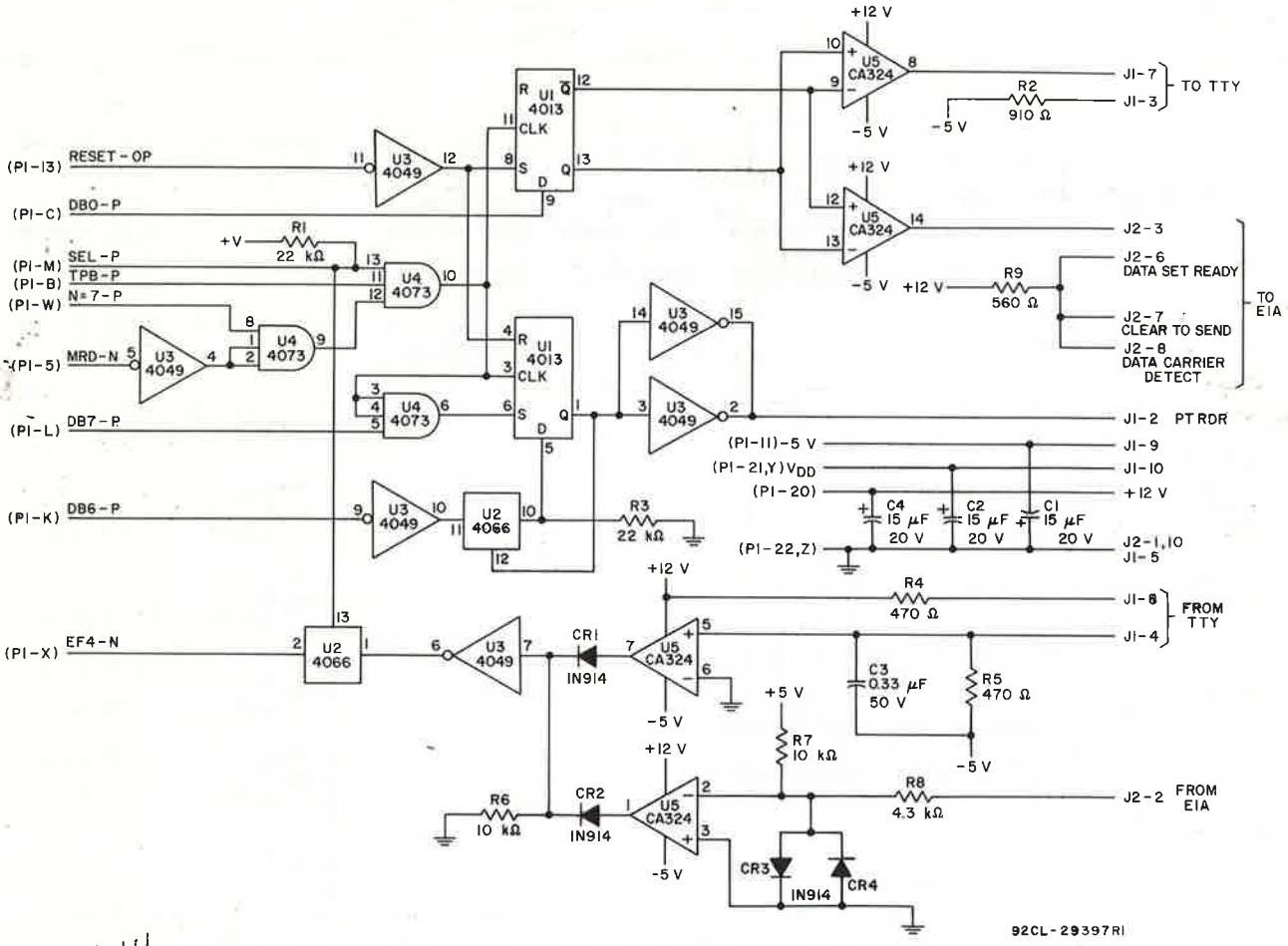


Fig. D12 — 4-kilobyte RAM module CDP18S205 layout diagram.

Parts List for Fig. D11 (CDP18S205)

C1 = 15 μF, ± 20%, 20 volts
C2, C3, C4, C5 = 0.1 μF, +80%, –20%, 50 volts
R1 through R8 = 22 kilohms, ± 5%, 1/4 watt
U1A, U2A = CDP1856D
U1B through U8B
U1C through U8C = CDP1822
U1D through U8D
U1E through U8E
U3A = CD4069BE
U5A, U6A = CDP1853D
U8A, U9A = CD4050BE



#14

Fig. D13 — Terminal interface module CDP18S507 logic and circuit diagram.

Parts List for Fig. D13 (CDP18S507)

C1, C2, C4 = 15 μF, ± 20%, 20 volts
C3 = 33 μF, ± 20%, 50 volts
CR1, CR2, CR3, CR4 = 1N914
J1, J2 = connector
R1, R3 = 22 kilohms, ± 5%, 1/4 watt
R2 = 910 ohms, ± 5%, 1/4 watt
R4, R5 = 470 ohms, ± 5%, 1/4 watt
R6, R7 = 10 kilohms, ± 5%, 1/4 watt
R8 = 4.3 kilohms, ± 5%, 1/4 watt
R9 = 560 ohms, ± 5%, 1/4 watt
U1 = CD4013BE
U2 = CD4066AE
U3 = CD4049BE
U4 = CD4073BE
U5 = CA324E

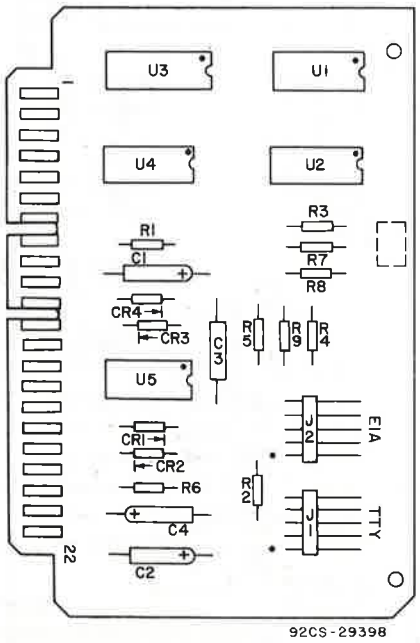


Fig. D14 — Terminal interface module CDP18S507 layout diagram.

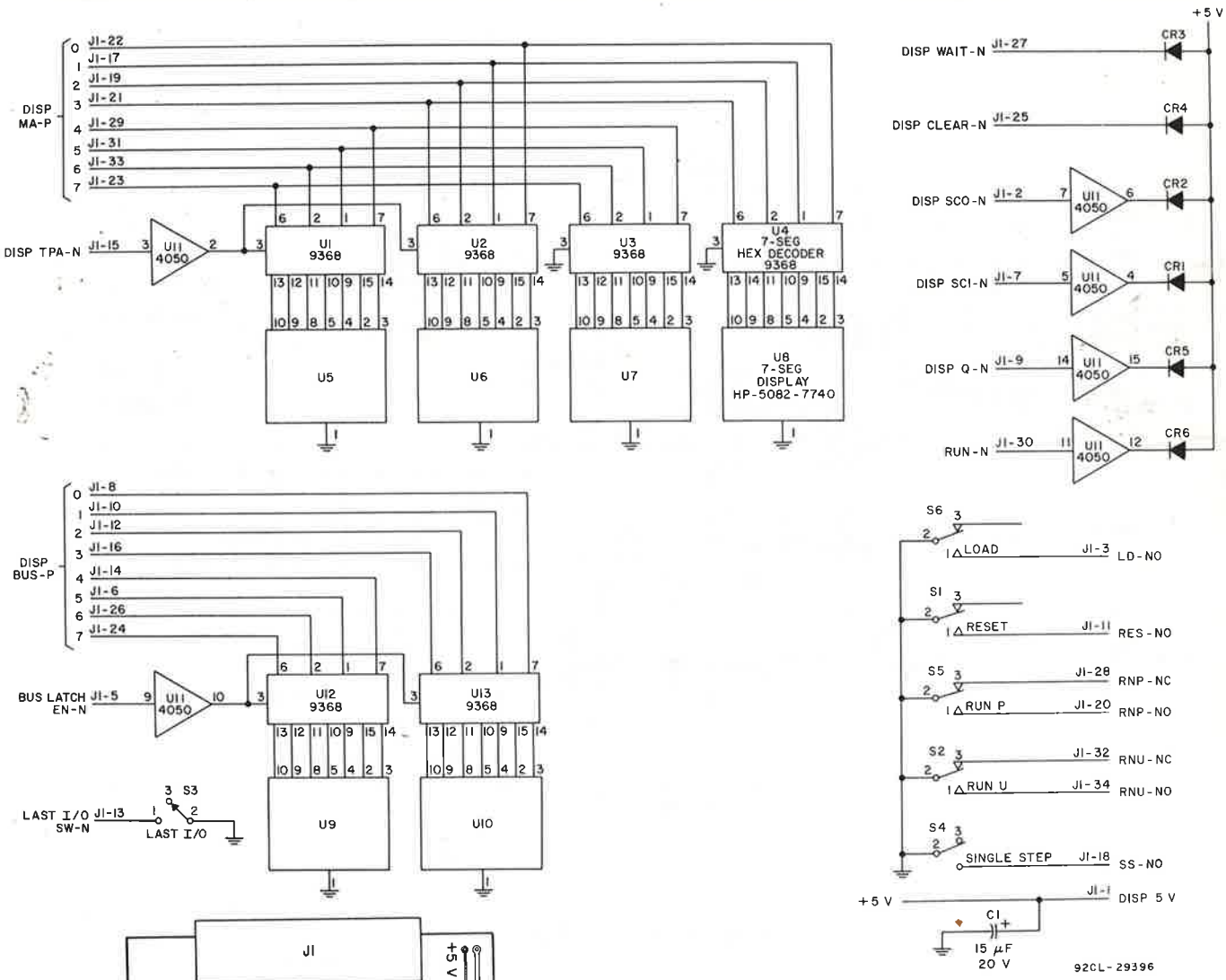


Fig. D15 — Display board logic and circuit diagram.

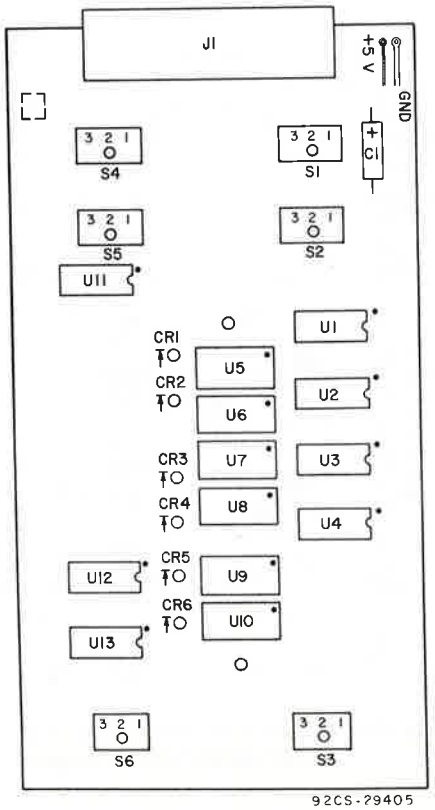


Fig. D16 — Display board layout diagram.

Parts List for Fig. D15

C1 = 15 μF, ± 20%, 20 volts
CR1, CR2, CR3, CR4, CR5, CR6 = light-emitting diodes
J1 = connector, 34 pin
S1, S2, S5, S6 = momentary action
S3, S4 = SPDT
U1, U2, U3, U4, U12, U13 = decoder driver
U5, U6, U7, U8, U9, U10 = 7-segment display
U11 = CD4050BE

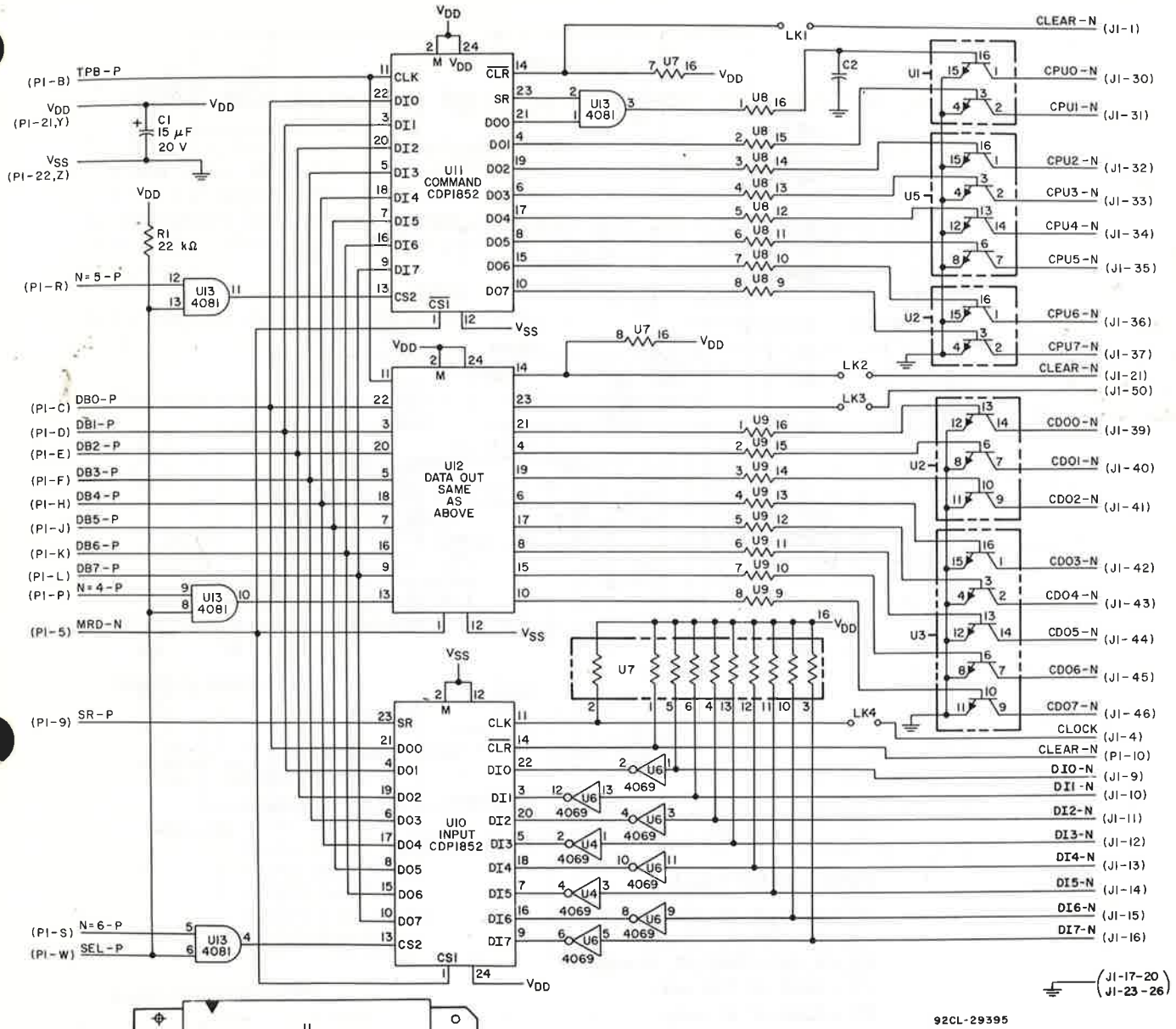
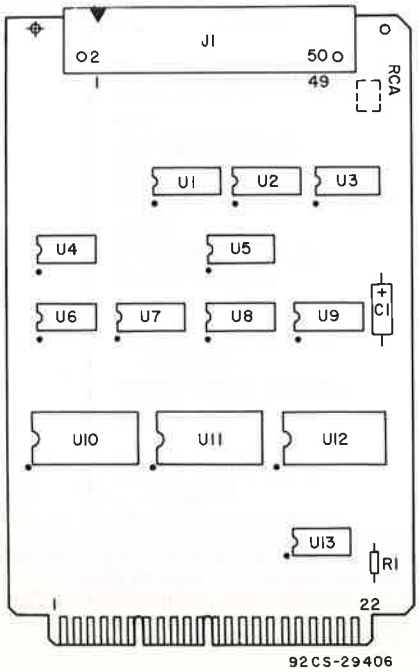


Fig. D17 — Disk interface module CDP18S813 logic and circuit diagram.



Parts List for Fig. D17

C1 = 15 μF, ± 20%, 20 volts
J1 = connector, 50 pin
R1 = 22 kilohms, ± 5%, 1/4 watt
U1, U2, U3, U5 = CA3083
U4, U6 = CD4069BE
U7 = resistor module, 3.3 kilohms
U8, U9 = resistor module, 2.2 kilohms
U10, U11, U12 = CDP1852D
U13 = CD4081BE

Note:
Cable connector should be aligned with arrow on J1.

Fig. D18 — Disk interface module CDP18S813 layout diagram.

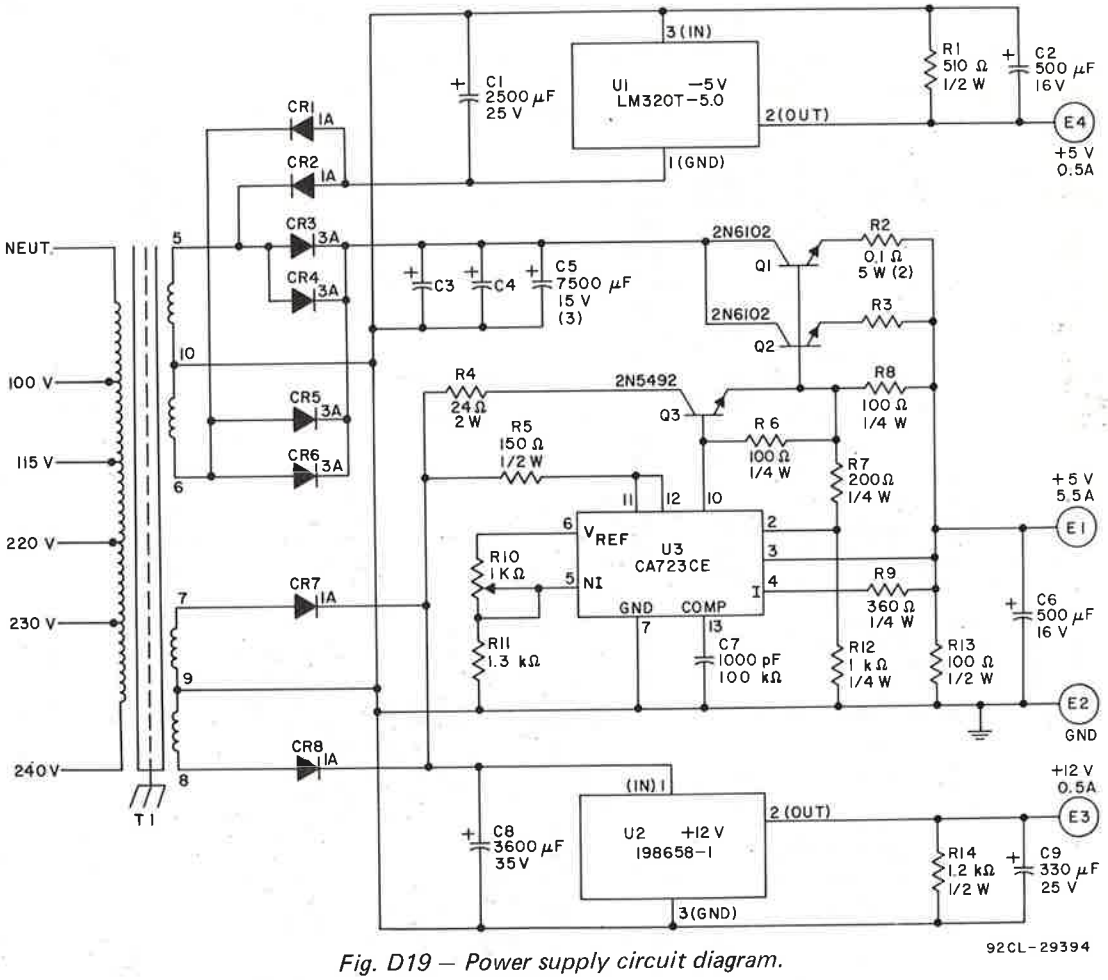


Fig. D19 — Power supply circuit diagram.

Parts List for Fig. D19

C1 = 2500 µF, 25 volts
C2, C6 = 500 µF, 16 volts
C3, C4, C5 = 7500 µF, 15 volts
C7 = 1000 pF, 100 volts
C8 = 3600 µF, 35 volts
C9 = 330 µF, 25 volts
CR1, CR2, CR7, CR8 = A14F, 1 A
CR3, CR4, CR5, CR6 = A15F, 3 A
Q1, Q2 = 2N6102
Q3 = 2N5492
R1 = 510 ohms, ± 5%, 1/2 watt
R2, R3 = 0.1 ohm, ± 10%, 5 watts
R4 = 24 ohms, ± 5%, 2 watts
R5 = 150 ohms, ±5%, 1/2 watt
R6, R8 =100 ohms, ± 5%, 1/4 watt
R7 = 220 ohms, ± 5%, 1/4 watt
R9 = 360 ohms, ± 5%, 1/2 watt
R10 = variable, 0-1000 ohms
R11 = 1300 ohms, ± 1%, 1/4 watt
R12 = 1000 ohms, ± 5%, 1/4 watt
R13 = 100 ohms, ± 5%, 1/2 watt
R14 = 1200 ohms, ± 5%, 1/2 watt
U1 = LM320T-5.0
U2 = 198658-1
U3 = CA723CE
T1 = Deltona #766-K29B, 50/60 Hz, input - 100 to 240 volts

Appendix E -
Instruction Summary for RCA CDP1802 COSMAC Microprocessor

The COSMAC instruction summary is given in the tabulations below. Hexadecimal notation is used to refer to the 4-bit binary codes.

In all registers bits are numbered from the least significant bit (LSB) to the most significant bit (MSB) starting with 0.

R(W): Register designated by W, where W=N or X, or P

R(W).0: Lower-order byte of R(W)
R(W).1: Higher-order byte of R(W)

Operation Notation
M(R(N)) → D;R(N)+1

This notation means: The memory byte pointed to by R(N) is loaded into D, and R(N) is incremented by 1.

INSTRUCTION SUMMARY
by Class of Operation

Register Operations

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
INCREMENT REG N	INC	1N	R(N) +1
DECREMENT REG N	DEC	2N	R(N) –1
INCREMENT REG X	IRX	60	R(X) +1
GET LOW REG N	GLO	8N	R(N).0→D
PUT LOW REG N	PLO	AN	D→R(N).0
GET HIGH REG N	GHI	9N	R(N).1→D
PUT HIGH REG N	PHI	BN	D→R(N).1

Memory Reference

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
LOAD VIA N	LDN	0N	M(R(N))→D; FOR N NOT 0
LOAD ADVANCE	LDA	4N	M(R(N))→D;R(N) +1
LOAD VIA X	LDX	F0	M(R(X))→D
LOAD VIA X AND ADVANCE	LDXA	72	M(R(X))→D; R(X) +1
LOAD IMMEDIATE	LDI	F8	M(R(P))→D; R(P) + 1
STORE VIA N	STR	5N	D→M(R(N))
STORE VIA X AND DECREMENT	STXD	73	D→M(R(X)); R(X) –1

Logic Operations♦♦

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
OR	OR	F1	M(R(X)) OR D→D
OR IMMEDIATE	ORI	F9	M(R(P)) OR D→D; R(P) +1
EXCLUSIVE OR	XOR	F3	M(R(X)) XOR D→D
EXCLUSIVE OR IMMEDIATE	XRI	FB	M(R(P)) XOR D→D; R(P) +1
AND	AND	F2	M(R(X)) AND D→D
AND IMMEDIATE	ANI	FA	M(R(P)) AND D→D; R(P) +1
SHIFT RIGHT	SHR	F6	SHIFT D RIGHT, LSB(D)→DF, 0→MSB(D)
SHIFT RIGHT WITH CARRY	SHRC	76♦	SHIFT D RIGHT, LSB(D)→DF, DF→MSB(D)
RING SHIFT RIGHT	RSHR		
SHIFT LEFT	SHL	FE	SHIFT D LEFT, MSB(D)→DF, 0→LSB(D)
SHIFT LEFT WITH CARRY	SHLC	7E♦	SHIFT D LEFT, MSB(D)→DF, DF→LSB(D)
RING SHIFT LEFT	RSHL		

♦NOTE: THIS INSTRUCTION IS ASSOCIATED WITH MORE THAN ONE MNEMONIC. EACH MNEMONIC IS INDIVIDUALLY LISTED.

♦♦NOTE: THE ARITHMETIC OPERATIONS AND THE SHIFT INSTRUCTIONS ARE THE ONLY INSTRUCTIONS THAT CAN ALTER THE DF.

Arithmetic Operations◆◆

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
ADD	ADD	F4	M(R(X)) +D→DF, D
ADD IMMEDIATE	ADI	FC	M(R(P) +D→DF, D; R(P) +1
ADD WITH CARRY	ADC	74	M(R(X)) +D +DF→DF, D
ADD WITH CARRY IMMEDIATE	ADCI	7C	M(R(P) +D +DF→DF, D R(P) +1
SUBTRACT D	SD	F5	M(R(X))−D→DF, D
SUBTRACT D IMMEDIATE	SDI	FD	M(R(P))−D→DF, D; R(P) +1
SUBTRACT D WITH BORROW	SDB	75	M(R(X))−D−(NOT DF)→DF; D
SUBTRACT D WITH BORROW, IMMEDIATE	SDBI	7D	M(R(P))−D−(NOT DF)→DF, D; R(P) +1
SUBTRACT MEMORY	SM	F7	D−M(R(X))→DF, D
SUBTRACT MEMORY IMMEDIATE	SMI	FF	D−M(R(P))→DF, D; R(P) +1
SUBTRACT MEMORY WITH BORROW	SMB	77	D−M(R(X))−(NOT DF)→DF, D
SUBTRACT MEMORY WITH BORROW, IMMEDIATE	SMBI	7F	D−M(R(P))−(NOT DF)→DF, D R(P) +1

Branch Instructions — Short Branch

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
SHORT BRANCH	BR	30	M(R(P))→R(P).0
NO SHORT BRANCH (SEE SKP)	NBR	38◆	R(P) +1
SHORT BRANCH IF D=0	BZ	32	IF D=0, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF D NOT 0	BNZ	3A	IF D NOT 0, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF DF = 1	BDF	33◆	IF DF=1, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF POS OR ZERO	BPZ		
SHORT BRANCH IF EQUAL OR GREATER	BGE		
SHORT BRANCH IF DF=0	BNF	3B◆	IF DF=0, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF MINUS	BM		
SHORT BRANCH IF LESS	BL		
SHORT BRANCH IF Q=1	BQ	31	IF Q=1, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF Q=0	BNQ	39	IF Q=0, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF1=1	B1	34	IF EF1=1, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF1=0	BN1	3C	IF EF1=0, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF2=1	B2	35	IF EF2=1, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF2=0	BN2	3D	IF EF2=0, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF3=1	B3	36	IF EF3=1, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF3=0	BN3	3E	IF EF3=0, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF4=1	B4	37	IF EF4=1, M(R(P))→R(P).0 ELSE R(P) +1
SHORT BRANCH IF EF4=0	BN4	3F	IF EF4=0, M(R(P))→R(P).0 ELSE R(P) +1

Branch Instructions — Long Branch

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
LONG BRANCH	LBR	C0	M(R(P))→R(P).1 M(R(P) +1)→R(P).0
NO LONG BRANCH (SEE LSKP)	NLBR	C8♦	R(P) +2
LONG BRANCH IF D=0	LBZ	C2	IF D=0, M(R(P))→R(P).1 M(R(P) +1)→R(P).0 ELSE R(P) +2
LONG BRANCH IF D NOT 0	LBNZ	CA	IF D NOT 0, M(R(P))→ R(P).1 M(R(P) +1)→ R(P).0 ELSE R(P) +2
LONG BRANCH IF DF=1	LBDF	C3	IF DF=1, M(R(P))→R(P).1 M(R(P) +1)→ R(P).0 ELSE R(P) +2
LONG BRANCH IF DF=0	LBNF	CB	IF DF=0, M(R(P))→R(P).1 M(R(P) +1)→ R(P).0 ELSE R(P) +2
LONG BRANCH IF Q=1	LBQ	C1	IF Q=1, M(R(P))→R(P).1 M(R(P) +1)→R(P).0 ELSE R(P) +2
LONG BRANCH IF Q=0	LBNQ	C9	IF Q=0, M(R(P))→R(P).1 M(R(P) +1)→ R(P).0 ELSE R(P) +2

Skip Instructions

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
SHORT SKIP (SEE NBR)	SKP	38♦	R(P) +1
LONG SKIP (SEE NLBR)	LSKP	C8♦	R(P) +2
LONG SKIP IF D=0	LSZ	CE	IF D=0, R(P) +2 ELSE CONTINUE
LONG SKIP IF D NOT 0	LSNZ	C6	IF D NOT 0, R(P) +2 ELSE CONTINUE
LONG SKIP IF DF=1	LSDF	CF	IF DF=1, R(P) +2 ELSE CONTINUE
LONG SKIP IF DF=0	LSNF	C7	IF DF=0, R(P) +2 ELSE CONTINUE
LONG SKIP IF Q=1	LSQ	CD	IF Q=1, R(P) +2 ELSE CONTINUE
LONG SKIP IF Q=0	LSNQ	C5	IF Q=0, R(P) +2 ELSE CONTINUE
LONG SKIP IF IE=1	LSIE	CC	IF IE=1, R(P) +2 ELSE CONTINUE

♦NOTE: THIS INSTRUCTION IS ASSOCIATED WITH MORE THAN ONE MNEMONIC. EACH MNEMONIC IS INDIVIDUALLY LISTED.

♦♦NOTE: THE ARITHMETIC OPERATIONS AND THE SHIFT INSTRUCTIONS ARE THE ONLY INSTRUCTIONS THAT CAN ALTER THE DF.

Control Instructions

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
IDLE	IDL	00	WAIT FOR DMA OR INTERRUPT; M(R(0))→BUS
NO OPERATION	NOP	C4	CONTINUE
SET P	SEP	DN	N→P
SET X	SEX	EN	N→X
SET Q	SEQ	7B	1→Q
RESET Q	REQ	7A	0→Q
SAVE	SAV	78	T→M(R(X))
PUSH X,P TO STACK	MARK	79	(X,P)→T; (X,P)→M(R(2)) THEN P→X; R(2)−1
RETURN	RET	70	M(R(X))→(X,P); R(X) +1 1→IE
DISABLE	DIS	71	M(R(X))→(X,P); R(X) +1 0→IE

Input–Output Byte Transfer

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
OUTPUT 1	OUT 1	61	M(R(X))→BUS; R(X) +1; N LINES = 1
OUTPUT 2	OUT 2	62	M(R(X))→BUS; R(X) +1; N LINES = 2
OUTPUT 3	OUT 3	63	M(R(X))→BUS; R(X) +1; N LINES = 3
OUTPUT 4	OUT 4	64	M(R(X))→BUS; R(X) +1; N LINES = 4
OUTPUT 5	OUT 5	65	M(R(X))→BUS; R(X) +1; N LINES = 5
OUTPUT 6	OUT 6	66	M(R(X))→BUS; R(X) +1; N LINES = 6
OUTPUT 7	OUT 7	67	M(R(X))→BUS; R(X) +1; N LINES = 7
INPUT 1	INP 1	69	BUS→M(R(X)); BUS→D; N LINES = 1
INPUT 2	INP 2	6A	BUS→M(R(X)); BUS→D; N LINES = 2
INPUT 3	INP 3	6B	BUS→M(R(X)); BUS→D; N LINES = 3
INPUT 4	INP 4	6C	BUS→M(R(X)); BUS→D; N LINES = 4
INPUT 5	INP 5	6D	BUS→M(R(X)); BUS→D; N LINES = 5
INPUT 6	INP 6	6E	BUS→M(R(X)); BUS→D; N LINES = 6
INPUT 7	INP 7	6F	BUS→M(R(X)); BUS→D; N LINES = 7

♦NOTE: THIS INSTRUCTION IS ASSOCIATED WITH MORE THAN ONE MNEMONIC. EACH MNEMONIC IS INDIVIDUALLY LISTED.

♦♦NOTE: THE ARITHMETIC OPERATIONS AND THE SHIFT INSTRUCTIONS ARE THE ONLY INSTRUCTIONS THAT CAN ALTER THE DF.

Appendix F -
ASCII - Hex Table

LEAST SIGNIFICANT HEX DIGIT	MOST SIGNIFICANT HEX DIGIT 0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	↑	n	~
F	SI	US	/	?	O	←	o	DEL

NOTES:

(1) Parity bit in most significant hex digit not included.
(2) Characters in columns 0 and 1 (as well as SP and DEL) are non-printing.
(3) Model 33 Teletypewriter prints codes in columns 6 and 7 as if they were column 4 and 5 codes.

Appendix G - UT20 Listing

```
!M
0000 ;                  0001
0000 ;                  0002 ..  UT20 IS A UTILITY PROGRAM USED TO ALTER
0000 ;                  0003 ..  MEMORY, DUMP MEMORY, AND BEGIN PROGRAM
0000 ;                  0004 ..  EXECUTION AT A GIVEN LOCATION.  THE COMMANDS
0000 ;                  0005 ..  ACCEPTED ARE $PHHHH (BEGIN EXECUTION AT THE
0000 ;                  0006 ..  SPECIFIED LOCATION WITH R0 AS PROGRAM
0000 ;                  0007 ..  COUNTER), !MHHHH DATA (PUT DATA AT SPECIFIED
0000 ;                  0008 ..  LOCATION), AND ?MHHHH HHHH (OUTPUT DATA
0000 ;                  0009 ..  FROM SPECIFIED LOCATION FOR SPECIFIC COUNT).
0000 ;                  0010 ..  AT THE BEGINNING OF A COMMAND ALL CHARACTERS
0000 ;                  0011 ..  ARE IGNORED UNTIL A ?, !, OR $ IS
0000 ;                  0012 ..  ENCOUNTERED.  IN THE ?M AND !M COMMANDS NON
0000 ;                  0013 ..  HEX CHARACTERS ARE IGNORED AFTER M UNTIL A
0000 ;                  0014 ..  HEX IS READ, THEN THE FIRST NON HEX
0000 ;                  0015 ..  CHARACTER MUST BE A SPACE.  NON HEX
0000 ;                  0016 ..  CHARACTERS BETWEEN HEX PAIRS OF THE DATA IN
0000 ;                  0017 ..  THE !M COMMAND ARE IGNORED EXCEPT FOR CR,
0000 ;                  0018 ..  SEMICOLON, AND COMMA.
0000 ;                  0019 ..  $L LOADS DATA (WRITTEN IN UT20 FORMAT) FROM
0000 ;                  0020 ..   FLOPPY DSK INTO MEMORY. THERE ARE 77 TRACKS
0000 ;                  0021 ..  AVAILABLE ON A DISKETTE (TRACK 0-76).
0000 ;                  0022 ..  LOADING STOPS IF THE EOF (DC3) IS DETECTED.
0000 ;                  0023 ..  THE BAUD RATE OF UT20 IS DEPENDENT UPON THE
0000 ;                  0024 ..  TERMINAL BEING USED.  A CR OR LF IS ENTERED
0000 ;                  0025 ..  AT THE BEGINNING TO SPECIFY THE APPROPRIATE
0000 ;                  0026 ..  DELAY BETWEEN BITS.  UT20 WILL ECHO
0000 ;                  0027 ..  CHARACTERS IF A CR IS CHOSEN AS THE
0000 ;                  0028 ..  TIMING CHARACTER.  ECHOING WILL NOT TAKE
0000 ;                  0029 ..  PLACE IF A LF IS INPUT AS THE TIMING
0000 ;                  0030 ..  CHARACTER.
0000 ;                  0031 ..  UT20, AT INITIATION, STORES ALL REGISTERS
0000 ;                  0032 ..  BETWEEN WRAM-32 AND WRAM IF IT FINDS RAM
0000 ;                  0033 ..  THERE (BUT R0, R1, AND R4.1 ARE CLOBBERED).
0000 ;                  0034 ..  ?R CAN BE USED TO TYPE THE CONTENTS OF THE 16
0000 ;                  0035 ..  REGISTERS (R0-RF). R0,R1,R4.1 WILL BE
0000 ;                  0036 ..  TYPED AS X'S (DON'T CARE).
0000 ;                  0037     PTER=#00    ..AUXILIARY FOR MAIN ROUTINE
0000 ;                  0038       CL=#01    ..CLOBBERED
0000 ;                  0039       ST=#02    ..STACK POINTER ONLY REFERENCE TO RAM
0000 ;                  0040      SUB=#03    ..SUBROUTINE PROGRAM COUNTER
0000 ;                  0041       PC=#05    ..MAIN PROGRAM COUNTER
0000 ;                  0042   SWITCH=CL     ..DISTINGUISHES BETWEEN ?M AND !M
0000 ;                  0043    DELAY=#0C    ..DELAY ROUTINE PROGRAM COUNTER
0000 ;                  0044      ASL=#0D    ..HEX ASSEMBLY REGISTER ON INPUT;
0000 ;                  0045                 ..AUX FOR HEX OUTPUT
0000 ;                  0046    CNTER=ASL    ..USED TO COUNT OUTPUT BYTES
0000 ;                  0047      AUX=#0E    ..AUX.1 HOLDS BIT-TIME CONSTANT
0000 ;                  0048     CHAR=#0F    ..CHAR.1 HOLDS I/O BYTE
0000 ;                  0049     WRAM=#8C1F  ..REGISTERS STORED IN RAM
0000 ;                  0050   LOADER=#8400  ..LOCATION LOADER PROGRAM
0000 ;                  0051 ..
0000 ;                  0052 ..  ENTER IN R0
0000 ;                  0053 ..
8000 ;                  0054          ORG#8000                  ..UT20 STARTS AT
8000 7100;              0055                                    ..M(8000)
8002 F880B0;            0056          DIS,#00                   ..P=X=0
8005 ;                  0057          LDI A.1(UT20) ;PHI R0     ..HOLDS HIGH BIT
8005 ;                  0058                                    ..AFTER FINGER OFF
                        0059 ..  MAY TRY TO GO TO 8000, NOT 0000
                        0060 ..  UNTIL FINGER IS OFF BUTTON
```

```
8005 ;                0061 ..
8005 ;                0062 ..  THE FOLLOWING WRITES REGISTER CONTENTS INTO
8005 ;                0063 ..  WRAM-32 THRU WRAM IF IT EXISTS. WRAM-34 IS
8005 ;                0064 ..  ASSUMED NOT TO BE RAM (ELSE ROUTINE OVERRUNS).
8005 ;                0065
8005 ;                0066 ..
8005 F88CB1;          0067          LDI A.1(WRAM) ;PHI CL    ..CL IS CLOB-
8008 ;                0068                                   ..BERED
8008 F81EA1;          0069          LDI A.0(WRAM-1) ;PLO CL  ..SET UP WHERE RF.0
800B ;                0070                                   ..IS TO GO, MINUS 1
800B F8A0B4;          0071          LDI #A0 ;PHI R4          ..R4.1 STORES A
800E ;                0072                                   ..MODIFIED INSTRUC.
800E E1;              0073          SEX CL
800F F8D051;          0074 LOOP2:   LDI #D0 ;STR CL          ..SET UP SEP INSTR.
8012 ;                0075                                   ..FOR RETURN
8012 F3;              0076          XOR                      ..CHECK IT WROTE
8013 3A29;            0077          BNZ UT20
8015 21;              0078          DEC CL                   ..PREPARE FOR MODI-
8016 ;                0079                                   ..FIED INSTRUCTION
8016 94FC70;          0080          GHI R4 ;ADI #70          ..IN THE 90'S?
8019 331D;            0081          BDF*+#04
801B FC21;            0082          ADI #21                  ..NO, 8N -> 9N
801D FC7F;            0083          ADI #7F                  ..YES, 9N -> 8(N-1)
801F B451;            0084          PHI R4 ;STR CL           ..SET MODIFIED
8021 ;                0085                                   ..INSTRUC INTO RAM
8021 D1;              0086          SEP CL                   ..EXECUTE INSTRUCS
8022 ;                0087                                   ..(80-9F)
8022 73;              0088          STXD                     ..STORE RESULT RAM
8023 21;              0089          DEC CL                   ..& BACK UP FOR
8024 94FB90;          0090          GHI R4   ;XRI#90       ..CK IF STORAGE DONE
8027 3A0F;            0091          BNZ LOOP2                 ..NEXT BYTE
8029 ;                0092 ..
8029 90B5B3;          0093 UT20:    GHI R0 ;PHI PC ;PHI SUB  ..#80->PC.1 & SUB.1
802C F830A5;          0094          LDI A.0(UT20A) ;PLO PC
802F D5;              0095          SEP PC
8030 E5;              0096 UT20A:   SEX PC
8031 7155;            0097          DIS,#55                  ..NOTE PC=5 ASSUMED
8033 6101;            0098          OUT 1,#01                ..SELECT RCA GROUP
8035 F88CB2;          0099          LDI A.1(WRAM)   ;PHI ST  ..SET STACK POINTER
8038 F800A2;          0100          LDI#00   ;PLO ST
803B ;                0101                                   ..TO M(8C00), ONLY
803B ;                0102                                   ..RAM USED
803B F8FEA3;          0103          LDI A.0(TIMALC) ;PLO SUB..READ ONE CHAR
803E ;                0104                                   ..TO SET TIMER
803E D3;              0105          SEP SUB
803F ;                0106 ..
803F ;                0107 ... INITIATION NOW DONE
803F ;                0108 ..
803F F89CA3;          0109 START:   LDI A.0(TYPE5D) ;PLO SUB
8042 F881B3;          0110          LDI A.1(TYPE5D)   ;PHI SUB
8045 D30D;            0111          SEP SUB; ,#0D            ..CR=CARRIAGE RET
8047 D30A;            0112 ST2:     SEP SUB; ,#0A            ..LF=LINE FEED
8049 D32A;            0113          SEP SUB; ,#2A            ..* PROMPT CHARAC
804B F800ADBD;        0114 IGNORE: LDI#00 ;PLO ASL;PHI ASL   ..PREPARE TO INPUT
804F ;                0115                                   ..HEX DIGITS,
804F ;                0116                                   ..CLEAR ASL
804F F83BA3;          0117          LDI A.0(READAH) ;PLO SUB
8052 D3;              0118          SEP SUB                  ..INPUT COMMAND
8053 FB24;            0119          XRI #24                  ..IS IT "$" ?
8055 C28207;          0120          LBZ DOLLAR
8058 FB05;            0121          XRI #05                  ..IS IT "!" ?
805A ;                0122                                   ..TEST $ XRI !
```

```
805A A1;          0123          PLO SWITCH                ..AND SAVE RESULT
805B CE;          0124          LSZ
805C FB1E;        0125          XRI #1E                   ..IS IT "?" ?
805E ;            0126                                    ..TES $ XRI ! XRI ?
805E 3A4B;        0127          BNZ IGNORE                ..IGNORE ALL UNTIL
8060 ;            0128                                    ..COMMAND IS READ
8060 ;            0129 ..
8060 ;            0130 ..  THE FOLLOWING IS COMMON FOR ?M AND !M
8060 ;            0131 ..  (SWITCH.0 = 0 FOR THE LATTER)
8060 ;            0132 ..
8060 D3;          0133 RDARGS: SEP SUB                    ..NOTE SUB AT
8061 ;            0134                                    ..READAH.  READ
8061 ;            0135                                    ..HEX ARGUMENTS
8061 FB4D;        0136          XRI #4D                   ..SHOULD BE "M"
8063 3ADC;        0137          BNZ ISITR                 ..CK FOR ?R
8065 D3;          0138 RD1:     SEP SUB
8066 3B65;        0139          BNF *-#01                 ..IGNORE NON HEX
8068 ;            0140                                    ..CHARS. AFTER "M"
8068 D3;          0141          SEP SUB
8069 3368;        0142          BDF *-#01                 ..READ FIRST ARG
806B ;            0143                                    ..(LOCA. IN MEMORY)
806B 9DB0;        0144          GHI ASL ;PHI PTER
806D 8DA0;        0145          GLO ASL ;PLO PTER         ..PTER NOW POINTS
806F ;            0146                                    ..TO USER MEMORY
806F F800ADBD;    0147          LDI#00 ;PLO ASL ;PHI ASL..CLEAR ASL
8073 1D;          0148          INC ASL                   ..?MXXXXCR PRINTS
8074 ;            0149                                    ..TWO HEX DIGITS
8074 9FFB0D;      0150          GHI RF    ;XRI#0D         ..CK FOR CR
8077 3A7E;        0151          BNZ TEST                  ..BR IF NOT A CR
8079 81;          0152          GLO SWITCH
807A 3A8D;        0153          BNZ LINE-#03              ..BR IF ?
807C 30E1;        0154          BR SYNERR                 ..OTHERWISE ERROR
807E FB2D;        0155 TEST:    XRI#2D                    ..CK FOR SPACE
8080 3AE1;        0156          BNZ SYNERR
8082 2D;          0157          DEC ASL                   ..ADJUST ASL
8083 81;          0158          GLO SWITCH                ..LOOK AT SWITCH
8084 32C6;        0159          BZ EX1                    ..IF 0 IT IS "!"
8086 ;            0160                                    ..OTHERWISE IT'S ?
8086 ;            0161 ..
8086 ;            0162 ..  THE FOLLOWING DOES (?M LOC COUNT) AND
8086 ;            0163 ..  (?MXXXXCR) COMMANDS
8086 D3;          0164 RD2:     SEP SUB
8087 3386;        0165          BDF RD2                   ..READ SECOND ARG
8089 ;            0166                                    ..(NUMBER OF BYTES)
8089 FB0D;        0167          XRI #0D                   ..NEXT CK FOR CR
808B 3AE1;        0168          BNZ SYNERR
808D F89CA3;      0169          LDI A.0(TYPE5D) ;PLO SUB..TYPE
8090 D30A;        0170 LINE:    SEP SUB; ,#0A             ..LF
8092 90BF;        0171          GHI PTER ;PHI CHAR        ..PREPARE LINE
8094 ;            0172                                    ..HEADING
8094 F8AEA3;      0173          LDI A.0(TYPE2) ;PLO SUB
8097 D3;          0174          SEP SUB                   ..TYPE 2 HEX DIGITS
8098 80BF;        0175          GLO PTER ;PHI CHAR
809A F8AEA3;      0176          LDI A.0(TYPE2) ;PLO SUB
809D D3;          0177          SEP SUB                   ..TYPE OTHER TWO
809E D320;        0178 TSPACE:  SEP SUB; ,#20             ..SPACE
80A0 ;            0179 ..
80A0 40BF;        0180 TLOOP:   LDA PTER ;PHI CHAR        ..FETCH ONE BYTE
80A2 ;            0181                                    ..FOR TYPING
80A2 F8AEA3;      0182          LDI A.0(TYPE2) ;PLO SUB
80A5 D3;          0183          SEP SUB                   ..TYPE 2 HEX
80A6 2D;          0184          DEC CNTER
```

```
80A7 8D;          0185          GLO CNTER
80A8 3AAD;        0186          BNZ TL3                 ..BRANCH NOT DONE
80AA 9D;          0187          GHI CNTER
80AB 323F;        0188          BZ START                ..BRANCH IF DONE
80AD 80FA0F;      0189 TL3:     GLO PTER ;ANI #0F       ..PTER DIV BY 16?
80B0 3AB8;        0190          BNZ TL2
80B2 D33B;        0191          SEP SUB; ,#3B           ..YES TYPE ";"
80B4 D30D;        0192          SEP SUB; ,#0D           ..THEN CR
80B6 3090;        0193          BR LINE
80B8 F6;          0194 TL2:     SHR                     ..DIV BY 2?
80B9 33A0;        0195          BDF TLOOP               ..NO, LOOP BACK
80BB 309E;        0196          BR TSPACE               ..ELSE TYPE SPACE &
80BD ;            0197                                  ..LOOP BACK
80BD ;            0198 ..
80BD ;            0199 ..  THE FOLLOWING DOES (!M LOC DATA) COMMAND
80BD ;            0200 ..  ENTER AT EX1
80BD ;            0201 ..
80BD ;            0202 ..  EFFECT OF THE FOLLOWING IS TO READ IN HEX
80BD ;            0203 ..  TERMINATING WITH A CR, IGNORING NON-HEX CHAR
80BD ;            0204 ..  PAIRS; EXCEPTIONS: A COMMA BEFORE A CR ALLOWS
80BD ;            0205 ..  THE INPUT TO CONTINUE ON THE NEXT LINE AND A
80BD ;            0206 ..  SEMICOLON ALLOWS THE !M COMMAND TO BE ASSUMED.
80BD ;            0207 ..
80BD D3;          0208 EX3:     SEP SUB                 ..INPUT UNTIL A
80BE ;            0209                                  ..HEX IS READ
80BE 3BBD;        0210          BNF EX3
80C0 D3;          0211 EX2:     SEP SUB                 ..LOOK FOR SECOND
80C1 ;            0212                                  ..HEX DIGIT
80C1 3BE1;        0213          BNF SYNERR              ..BR IF NOT HEX
80C3 8D50;        0214          GLO ASL ;STR PTER       ..**SET BYTE**
80C5 10;          0215          INC PTER
80C6 D3;          0216 EX1:     SEP SUB                 ..NOTE SUB @ READAH
80C7 33C0;        0217          BDF EX2                 ..BRANCH IF HEX
80C9 FB0D;        0218          XRI #0D                 ..CHECK IF CR
80CB 323F;        0219          BZ START
80CD FB21;        0220 EX4:     XRI #21                 ..ELSE CK FOR COMMA
80CF ;            0221                                  ..(TEST CR XRI ",")
80CF 32BD;        0222          BZ EX3                  ..IF ELSE BRANCH
80D1 FB17;        0223          XRI #17                 ..ELSE CK FOR ";"
80D3 ;            0224                                  ..(TEST CR XRI
80D3 ;            0225                                  ..","  XRI ";")
80D3 3AC6;        0226          BNZ EX1                 ..IGNORE ALL ELSE
80D5 D3;          0227          SEP SUB                 ..ON ";" IGNORE ALL
80D6 ;            0228                                  ..UNTIL CR, THEN
80D6 ;            0229                                  ..LOOP BACK
80D6 FB0D;        0230          XRI #0D
80D8 3AD5;        0231          BNZ *-03
80DA 3065;        0232          BR RD1                  ..THEN BRANCH BACK
80DC ;            0233                                  ..FOR !M COMMAND
80DC FB1F;        0234 ISITR:   XRI#1F                  ..IS IT R?
80DE C282E8;      0235          LBZ TYPER               ..BR IF R
80E1 ;            0236 ..
80E1 F89CA3;      0237 SYNERR: LDI A.0(TYPE5D);PLO SUB  ..GENERAL RESULT
80E4 ;            0238                                  ..SYNTACTIC ERROR
80E4 D30D;        0239          SEP SUB; ,#0D           ..CR
80E6 C08200;      0240          LBR FSYNER
80E9 ;            0241 ..
80E9 ;            0242 ..
80E9 ;            0243 ..
80E9 ;            0244 ..  SUBROUTINES
80E9 ;            0245 ..
80E9 ;            0246 ..
```

```
80E9 ;              0247          ORG*+#01
80EA ;              0248 ..   DELAY ROUTINE
80EA ;              0249 ..   DELAY IS 2(1+AUX.1(3+@SUB))
80EA ;              0250 ..   USED BY TYPE, READ, AND TIMALC.
80EA ;              0251 ..   AUX.1 IS ASSUMED TO HOLD A DELAY CONSTANT
80EA ;              0252 ..   =((BIT TIME OF TERMINAL)/
80EA ;              0253 ..   (20*INSTR TIME OF COSMAC))-1.
80EA ;              0254 ..   THIS CONSTANT CAN BE GENERATED
80EA ;              0255 ..   AUTOMATICALLY BY THE TIMALC ROUTINE.
80EA ;              0256 ..
80EA DCDCDCDC;      0257 DEXIT:   SEP RC;SEP RC;SEP RC;SEP RC      ..4 NOP'S
80EE D3;            0258          SEP SUB           ..RETURN
80EF 9EF6AE;        0259 DELAY1: GHI AUX ;SHR ;PLO AUX    ..SHIFT OUT
80F2 ;              0260                                  ..ECHO FLAG
80F2 2E;            0261 DELAY2: DEC AUX                  ..AUX.0 HOLDS BASIC
80F3 ;              0262                                  ..BIT DELAY
80F3 43FF01;        0263          LDA SUB ;SMI #01        ..PICK UP CONSTANT
80F6 3AF4;          0264          BNZ *-#02               ..LOOP AS SPECIFIED
80F8 ;              0265                                  ..BY CALL
80F8 8E;            0266          GLO AUX                 ..DONE YET?
80F9 32EA;          0267          BZ DEXIT
80FB 23;            0268          DEC SUB                 ..POINTS SUB AT
80FC ;              0269                                  ..DELAY POINTER
80FC 30F2;          0270          BR DELAY2
80FE ;              0271 ..
80FE ;              0272 ..   ROUTINE TO CALCULATE BYTE TIME AND ECHO
80FE ;              0273 ..   FLAG.  WAITS FOR LF(NO ECHO) OR CR(ECHO)
80FE ;              0274 ..   TO BE TYPED IN.  ALSO SETS UP POINTER TO
80FE ;              0275 ..   DELAY ROUTINE.
80FE ;              0276 ..   AUX.1 ENDS UP HOLDING, IN THE MOST
80FE ;              0277 ..   SIGNIFICANT 7 BITS, THE DELAY CONSTANT.
80FE ;              0278 ..   LEAST SIGNIFICANT BIT IS ZERO FOR ECHO,
80FE ;              0279 ..   ONE FOR NO ECHO.
80FE ;              0280 ..
80FE 93BC;          0281 TIMALC: GHI SUB ;PHI DELAY
8100 F8EFAC;        0282          LDI A.0(DELAY1) ;PLO DELAY
8103 F800AEAF;      0283          LDI #00 ;PLO AUX ;PLO CHAR
8107 ;              0284                                  ..DELAY ROUT. READY
8107 3707;          0285          B4*                     ..WAIT START BIT
8109 3F09;          0286          BN4*                    ..WAIT FOR FIRST
810B ;              0287                                  ..NONZERO DATA BIT
810B F803;          0288          LDI #03                 ..SET UP FOR 10
810D ;              0289                                  ..EXECUTIONS SO
810D ;              0290                                  ..ROUND-OFF MINIMAL
810D FF01;          0291 TC2:     SMI #01
810F 3A0D;          0292          BNZ *-#02
8111 8F;            0293          GLO CHAR                ..LOOK TO SEE IF
8112 ;              0294                                  ..DATA CHANGED
8112 ;              0295                                  ..PREVIOUSLY
8112 3A17;          0296          BNZ ZRONE               ..BR IF IT 6HAD
8114 3719;          0297          B4 INCR                 ..ELSE LOOK FOR
8116 ;              0298                                  ..CHANGE TO ZERO
8116 ;              0299                                  ..BRANCH IF NO
8116 1F;            0300          INC CHAR                ..YES, SET SWITCH
8117 371E;          0300 ZRONE:   B4 DAUX                 ..LOOK FOR CHANGE
8119 ;              0301                                  ..TO 1, BR IF YES
8119 1E;            0302 INCR:    INC AUX
811A F807;          0303          LDI #07                 ..SET UP FOR 20
811C ;              0304                                  ..INSTRUCTION LOOPS
811C 300D;          0305          BR TC2
811E ;              0306 ..
811E ;              0307 ..   AUX.0 NOW HOLDS #LOOPS IN 2 BIT TIMES
811E ;              0308 ..
```

```
811E 2E2E;          0309 DAUX:   DEC AUX ;DEC AUX          ..REDUCE COUNT TO
8120 ;              0310                                   ..BALANCE FIXED
8120 ;              0311                                   ..OVERLOAD IN
8120 ;              0312                                   ..CALLING DELAY
8120 8EF901BE;      0313         GLO AUX ;ORI #01 ;PHI AUX..LSB AUX.1 = 1.5
8124 DC0C;          0314         SEP RC; ,#0C              ..BIT TIME DELAY
8126 3F2C;          0315         BN4 WAIT                  ..BR IF LF(NO ECHO)
8128 ;              0316                                   ..LSB AUX.1=1
8128 9EFAFE;        0317         GHI AUX ;ANI#FE
812B BE;            0318         PHI AUX                   ..CR(ECHO)
812C ;              0319                                   ..LSB AUX.1=0
812C DC26;          0320 WAIT:   SEP RC; ,#26
812E D5;            0321         SEP R5
812F ;              0322 ..
812F ;              0323 ..
812F ;              0324 ..  READ ROUTINE--READS 1 BYTE INTO CHAR.1. WHEN
812F ;              0325 ..  ENTERED VIA READAH, THEN IF INPUT IS A HEX
812F ;              0326 ..  DIGIT ITS HEX VALUE IS SHIFTED INTO ASL FROM
812F ;              0327 ..  THE RIGHT AND DF=1, ELSE DF=0; CLOBBERS CHAR,
812F ;              0328 ..  AUX.0, (ASL ON READAH).  LEAVES BYTE IN D
812F ;              0329 ..  (BUT CLOBBERED IF SUBR LINKAGE IS USED).
812F ;              0330 ..  LEAVES PC AT READAH ENTRY POINT; EXITS TO R5.
812F ;              0331 ..
812F ;              0332 ..
812F ;              0333 ..     WARNING: READ PROCESS HAS NOT FINISHED.  DO
812F ;              0334 ..     NOT TYPE IMMEDIATELY, OR ELSE ENTER TYPE VIA
812F ;              0335 ..     TYPE5D.
812F ;              0336 ..
812F ;              0337 ..
812F FC07;          0338 CKDEC:  ADI #07                   ..CK FOR ASCII
8131 ;              0339                                   ..DECIMAL INPUT
8131 3337;          0340         BDF NFND
8133 FC0A;          0341         ADI #0A
8135 3377;          0342         BDF FND                   ..SUB NET 30
8137 FC00;          0343 NFND:   ADI #00                   ..SETS DF=0
8139 9F;            0344 REXIT:  GHI CHAR                  ..CHARACTER INTO D
813A D5;            0345         SEP R5
813B F800;          0346 READAH: LDI #00
813D 38;            0347         SKP                       ..SKIP OVER
813E ;              0348                                   ..TO READ1
813E 93;            0349 READ:   GHI SUB                   ..CONSTANT WITH
813F ;              0350                                   ..A VALUE > 0
813F AF;            0351 READ1:  PLO CHAR                  ..SET ENTRY FLAG
8140 F880BF;        0352 READ2:  LDI #80 ;PHI CHAR         ..INITIALIZE INPUT
8143 ;              0353                                   ..BYTE-WHEN SHIFTED
8143 ;              0354                                   ..80 IS 1, WILL BE
8143 ;              0355                                   ..DONE
8143 E2;            0356         SEX ST
8144 3F44;          0357         BN4 *                     ..WAIT FOR END OF
8146 ;              0358                                   ..LAST DATA BIT
8146 3746;          0359         B4 *                      ..WAIT FOR PRESENT
8148 ;              0360                                   ..START BIT
8148 DC02;          0361         SEP RC; ,#02              ..DELAY HALF
814A ;              0362                                   ..BIT TIME
814A ;              0363 ..
814A F80052;        0364 NOBIT:  LDI #00 ;STR ST
814D 9EFA01;        0365 LOOP5:  GHI AUX ;ANI #01          ..CHECK IF ECHO
8150 ;              0366                                   ..INDICATOR IS
8150 ;              0367                                   ..LSB OF AUX.1
8150 F152;          0368         OR ;STR ST                ..OUTPUT IS ONE(NO
8152 ;              0369                                   ..EFFECT) ON NOECHO
8152 6722;          0370         OUT 7 ;DEC ST
```

```
8154 ;                  0371 ..
8154 DC07;              0372 LOOP5B: SEP RC; ,#07              ..DELAY ONE
8156 ;                  0373                                   ..BIT TIME
8156 F80152;            0374         LDI #01 ;STR ST
8159 9FF6BF;            0375         GHI CHAR ;SHR ;PHI CHAR ..SHIFT INPUT CHAR.
815C 3365;              0376         BDF NEXT                  ..BR IF INPUT
815E ;                  0377                                   ..FINISHED D=CHAR.1
815E F980;              0378         ORI #80
8160 3F4A;              0379         BN4 NOBIT                 ..BR IF INPUT
8162 ;                  0380                                   ..BIT A ZERO
8162 BF;                0381         PHI CHAR                  ..ELSE PUT OK'D
8163 ;                  0382                                   ..VALUE AWAY
8163 304D;              0383         BR LOOP5
8165 ;                  0384 ..
8165 ;                  0385 ..  NOW HAVE BYTE READ INTO CHAR.1
8165 ;                  0386 ..
8165 6722;              0387 NEXT:   OUT 7; DEC ST             ..OUTPUT STOP BIT
8167 3240;              0388         BZ READ2                  ..BR IF D=0,
8169 ;                  0389                                   ..CHAR.1 IS A NULL
8169 8F;                0390         GLO CHAR                  ..CHECK ENTRY FLAG
816A 3A39;              0391         BNZ REXIT                 ..BR IF ENTRY VIA
816C ;                  0392                                   ..READ
816C 9F;                0393 CKHXE:  GHI CHAR
816D FF41;              0394         SMI #41                   ..CK FOR ASCII HEX
816F 3B2F;              0395         BNF CKDEC                 ..AT TOP OF ROUTINE
8171 FF06;              0396         SMI #06                   ..CK FOR A THRU F
8173 3337;              0397         BDF NFND
8175 FC10;              0398         ADI #10                   ..SUB NET 37
8177 ;                  0399 ..
8177 AE;                0400 FND:    PLO AUX                   ..SAVE TO SHIFT
8178 ;                  0401                                   ..INTO ASL
8178 9D;                0402         GHI ASL
8179 FEFEFEFE;          0403         SHL ;SHL ;SHL ;SHL        ..SHIFT ASL.1
817D ;                  0404                                   ..LEFT FOUR
817D 52;                0405         STR ST
817E 8D;                0406         GLO ASL
817F F6F6F6F6;          0407         SHR ;SHR ;SHR ;SHR        ..SHIFT ASL.0 RT 4
8183 F1BD;              0408         OR ;PHI ASL               ..COMBINE
8185 8D;                0409         GLO ASL
8186 FEFEFEFE;          0410         SHL ;SHL ;SHL ;SHL        ..SHIFT ASL.0
818A ;                  0411                                   ..LEFT FOUR
818A 52;                0412         STR ST
818B 8EFA0FF1AD;        0413         GLO AUX ;ANI #0F ;OR ;PLO ASL ..COMBINE
8190 FF00;              0414         SMI #00                   ..SET DF
8192 3039;              0415         BR REXIT
8194 ;                  0416 ..
8194 ;                  0417 ..  TYPE ROUTINE -- TYPES 1 BYTE FROM @R5!, @R6!,
8194 ;                  0418 ..  OR CHAR.1, OR TYPES A BYTE AS 2 HEX DIGITS FROM
8194 ;                  0419 ..  CHAR.1 FOLLOWS A LINE FEED BY SIX NULLS.
8194 ;                  0420 ..  USES 2 AUXILIARY REGS - AUX AND CHAR - PLUS
8194 ;                  0421 ..  RAM LOCATION @ST.  EXITS READY TO TYPE 1 BYTE
8194 ;                  0422 ..  FROM @R5!.  EXITS TO R5 WHEN ENTERED AT TYPE5D,
8194 ;                  0423 ..  PAUSES TO ALLOW AN EARLIER READ TO COMPLETE.
8194 ;                  0424 ..
8194 ;                  0425 ..  AUX.0 HOLDS OUTPUT CHAR (AT FIRST), THEN THE
8194 ;                  0426 ..  DELAY CONSTANT BETWEEN BITS.  CHAR.0 HOLDS THE
8194 ;                  0427 ..  NUMBER OF BITS (11) IN ITS LOWER DIGIT, AND
8194 ;                  0428 ..  IN ITS UPPER DIGIT HOLDS A CODE --
8194 ;                  0429 ..          0 FOR BYTE OUTPUT
8194 ;                  0430 ..          1 FOR FIRST HEX OUTPUT
8194 ;                  0431 ..          2 FOR LAST NULL OUTPUT
8194 ;                  0432 ..          8 FOR LF OUTPUT
```

```
8194 ;                0433 ..
8194 ;                0434 ..
8194 ;                0435          ORG #819C
819C DC17;            0436 TYPE5D:  SEP RC; ,#17          ..3 BIT TIME DELAY
819E 38;              0437          SKP                   ..SKIP TO TYPE5
819F D5;              0438 TEXIT:   SEP R5
81A0 4538;            0439 TYPE5:   LDA R5 ;SKP           ..ENTRY FOR UT20
81A2 ;                0440                                ..SKIP TO TYPE
81A2 4638;            0441 TYPE6:   LDA R6 ;SKP           ..ENTRY FOR G.P.
81A4 ;                0442                                ..IMMED,TH
81A4 9F;              0443 TYPE:    GHI CHAR
81A5 AE;              0444 TY1:     PLO AUX               ..SAVE BYTE
81A6 FB0A;            0445          XRI#0A                ..IS IT LINE FEED?
81A8 3AC0;            0446          BNZ TY2
81AA F88B;            0447          LDI#8B                ..(# BITS)+(# NULLS
81AC ;                0448                                ..TO FOLLOW LF + 1)
81AC 30C2;            0449          BR TY3
81AE 9F;              0450 TYPE2:   GHI CHAR              ..UT20 ENTRY
81AF F6F6F6F6;        0451 TY4:     SHR ;SHR ;SHR ;SHR    ..SHIFT FIRST HEX
81B3 ;                0452                                ..TO THE RIGHT
81B3 FCF6;            0453          ADI#F6                ..CONVERT TO HEX
81B5 3BB9;            0454          BNF *+#04             ..IF "A" OR MORE
81B7 FC07;            0455          ADI#07                ..ADD NET 37
81B9 FFC6AE;          0456          SMI#C6 ;PLO AUX       ..ELSE ADD NET 30
81BC F81B;            0457          LDI#1B                ..10+(# OF BITS)
81BE 30C2;            0458          BR TY3
81C0 ;                0459 ..
81C0 F80B;            0460 TY2:     LDI#0B                ..#BITS TO OUTPUT
81C2 AF;              0461 TY3:     PLO CHAR              ..SAVE MAIN TALLY
81C3 ;                0462                                ..VALUE
81C3 E2;              0463          SEX ST
81C4 ;                0464 ..
81C4 F80052;          0465 BEGIN:   LDI#00 ;STR ST        ..FOR START BIT
81C7 67;              0466          OUT 7
81C8 22;              0467          DEC ST                ..BACK TO WHERE
81C9 ;                0468                                ..IT WAS
81C9 8E;              0469          GLO AUX               ..PUT CHAR BACK
81CA 52;              0470 PREBIT:  STR ST
81CB DC07;            0471 BITS:    SEP RC; ,#07          ..DELAY 1 BIT TIME
81CD 2F;              0472          DEC CHAR              ..DECREMENT TALLY
81CE F0AEFA0152;      0473          LDX ;PLO AUX ;ANI#01 ;STR ST
81D3 67;              0474          OUT 7                 ..OUTPUT DATA BIT
81D4 22;              0475          DEC ST
81D5 8FFA0F;          0476          GLO CHAR ;ANI#0F
81D8 32E28E;          0477          BZ NXCHAR ;GLO AUX    ..AUX.0 TO STRETCH
81DB ;                0478                                ..DELAY
81DB 8EF6F980;        0479          GLO AUX ;SHR ;ORI#80  ..SHIFT TO
81DF 52;              0480          STR ST                ..NEXT BIT
81E0 30CA;            0481          BR PREBIT
81E2 ;                0482 ..
81E2 8FFCFB;          0483 NXCHAR:  GLO CHAR ;ADI#FB      ..SET UP FOR
81E5 AF;              0484          PLO CHAR              ..NEXT CHAR
81E6 3B9F;            0485          BNF TEXIT             ..EXIT IF NO MORE
81E8 FF1B;            0486          SMI#1B                ..TEST FOR
81EA ;                0487                                ..ALTERNATIVES
81EA 329F;            0488          BZ TEXIT              ..IF JUST TYPED
81EC ;                0489                                ..LAST NULL
81EC 3BF2;            0490          BNF HEX2              ..IF JUST TYPED
81EE ;                0491                                ..FIRST NULL, LF
81EE ;                0492                                ..OR NULL
81EE F800;            0493          LDI#00                ..PREPARE TO TYPE
81F0 ;                0494                                ..NULL
```

```
81F0 30FD;          0495          BR HX22
81F2 ;              0496 ..
81F2 9FFA0F;        0497 HEX2:    GHI CHAR ;ANI#0F          ..GET SECOND HEX
81F5 ;              0498                                    ..DIGIT
81F5 FCF6;          0499          ADI#F6                    ..CONVERT TO HEX
81F7 3BFB;          0500          BNF *+#04                 ..IF "A" OR MORE
81F9 FC07;          0501          ADI#07                    ..ADD NET 37
81FB FFC6;          0502          SMI#C6                    ..ELSE ALL NET 30
81FD AE;            0503 HX22:    PLO AUX                   ..STORE CHAR AWAY
81FE 30C4;          0504          BR BEGIN
8200 ;              0505 ..
8200 ;              0506 ..
8200 D30A;          0507 FSYNER:  SEP SUB; ,#0A             ..LF
8202 D33F;          0508          SEP SUB; ,#3F             ..?
8204 C0803F;        0509          LBR START
8207 ;              0510 ..
8207 ;              0511 ..  THE FOLLOWING DOES $P HHHH ,$U HHHH
8207 ;              0512 ..
8207 F85FA1;        0513  DOLLAR:LDI A.0(INTRPT)    ;PLO R1 ..R1 IS POINTING
820A F882B1;        0514          LDI A.1(INTRPT)   ;PHI R1 ..AT INTRPT
820D D3;            0515          SEP SUB                   ..SUB.0=READAH
820E FB55;          0516          XRI #55                   ..CHECK FOR "U"
8210 3231;          0517          BZ D1                     ..CON'T WITH "U"
8212 FB19;          0518          XRI #19                   ..CHECK FOR "L"
8214 3245;          0519          BZ DOLL                   ..IF "L"
8216 FB1C;          0520          XRI#1C                    ..CK FOR "P"
8218 CA80E1;        0521          LBNZ SYNERR               ..NOT P EITHER
821B D3;            0522          SEP SUB
821C 331B;          0523          BDF *-#01                    ..ASSEMBLE HEX
821E ;              0524                                    ..STRING INTO ASL
821E FB0D;          0525          XRI #0D                   ..FIRST NON-HEX
8220 ;              0526                                    ..MUST BE CR
8220 CA80E1;        0527          LBNZ SYNERR
8223 F89CA3;        0528          LDI A.0(TYPE5D) ;PLO SUB
8226 D30A;          0529          SEP SUB; ,#0A             ..LF
8228 E5;            0530          SEX PC
8229 7055;          0531          RET,#55
822B 6100;          0532          OUT 1,#00                 ..CLEAR I/O DECODER
822D 6704;          0533          OUT 7,#04                 ..BIT 2 DESELECTS
822F ;              0534                                    ..THE 2 LEVEL I/O
822F 3039;          0535          BR D2
8231 D3;            0536 D1:      SEP SUB
8232 3331;          0537          BDF D1                    ..ASSEMBLE HEX
8234 ;              0538                                    ..STRING INTO ASL
8234 FB0D;          0539          XRI #0D                   ..FIRST NON-HEX
8236 ;              0540                                    ..MUST BE CR
8236 CA80E1;        0541          LBNZ SYNERR
8239 9DB0;          0542  D2:     GHI ASL ;PHI R0
823B 8DA0;          0543          GLO ASL ;PLO R0           ..SET UP NEXT PC
823D F89CA3;        0544          LDI A.0(TYPE5D) ;PLO SUB
8240 D30A;          0545          SEP SUB; ,#0A             ..LF
8242 E5;            0546          SEX PC
8243 7000;          0547          RET, #00                  ..AND USER PROGRAM
8245 ;              0548                                    ..BEGINS (IN R0)
8245 ;              0549                                    ..EXIT TO UT20
8245 ;              0550 ..  THE FOLLOWING DOES $L
8245 F800A0;        0551  DOLL:LDI A.0(LOADER)    ;PLO R0
8248 F884B0;        0552          LDI A.1(LOADER)   ;PHI R0
824B E0;            0553          SEX R0
824C D0;            0554          SEP R0
824D ;              0555 ..  MSGE ROUTINE
824D ;              0556 ..  THIS ROUTINE INITIALIZES RC TO
824D ;              0557 ..  POINT AT THE DELAY ROUTINE.
```

```
824D ;                  0558 ..  IT TYPES OUT DATA POINTED TO BY R6. THIS
824D ;                  0559 ..  ROUTINE USES THE STANDARD CALL AND RET ROUTINES.
824D F8EFAC;            0560  MSGE:LDI A.0(DELAY1)   ;PLO RC
8250 F880BC;            0561         LDI A.1(DELAY1)   ;PHI RC
8253 DC12;              0562         SEP RC,#12                ..DELAY
8255 46BF;              0563  STRNG:LDA R6   ;PHI RF           ..LOAD CHAR TO RF.1
8257 325E;              0564         BZ EXIT1
8259 D481A4;            0565         SEP R4; ,A(TYPE)          ..TYPE OUT CHAR
825C 3055;              0566         BR STRNG
825E D5;                0567  EXIT1:SEP R5
825F ;                  0568 ..  INTERRUPT ROUTINE
825F ;                  0569 ..  IT INITIALIZES R4,R5 TO POINT AT
825F ;                  0570 ..  THE CALL AND RETURN ROUTINES. IT CALLS OSTRNG,
825F ;                  0571 ..  AND OUTPUT 'INTRPT ON' MESSAGE.
825F ;                  0572 ..  IT EXITS OSTRNG WITH R3 AS PROGRAM COUNTER,
825F ;                  0573 ..  THEN IT TRANSFERS CONTROL TO UT20.
825F F864A4;            0574  INTRPT:LDI A.0(CALL)   ;PLO R4   ..INITIALIZE CALL
8262 ;                  0575                                    ..POINTER
8262 F874A5;            0576         LDI A.0(RET)   ;PLO R5    ..INITIALIZE RET
8265 ;                  0577                                    ..POINTER
8265 F883B4;            0578         LDI A.1(CALL)   ;PHI R4   ..CALL AND RET ON
8268 B5;                0579         PHI R5                    ..SAME PAGE
8269 F81FA2;            0580         LDI#1F   ;PLO R2          ..INITIALIZE I/O
826C F88CB2;            0581         LDI A.1(WRAM)   ;PHI R2   ..POINTER
826F F876A3;            0582         LDI A.0(MSG)   ;PLO R3    ..INITIALIZE PC
8272 F882B3;            0583         LDI A.1(MSG)   ;PHI R3
8275 D3;                0584         SEP R3
8276 D4824D;            0585  MSG:SEP R4; ,A(MSGE)
8279 494E5452505421;0586         ,T'INTRPT!',#00
8280 00;                0586
8281 ;                  0587 ..  ENTER ROUTINE
8281 ;                  0588 ..  THIS ROUTINE INITIALIZES RC TO POINT AT
8281 ;                  0589 ..  THE DELAY ROUTINE. IT ALSO INITIALIZES R2 TO
8281 ;                  0590 ..  LOC #8C00 (THE I/O LOCATION USED BY UT20).
8281 ;                  0591 ..  IT DISABLES INTERRUPT, SELECTS RCA I/O
8281 ;                  0592 ..  GROUP, AND TRANSFERS CONTROL TO UT20.
8281 E3;                0593  ENTER:SEX R3                      ..X=P=3
8282 F8EFAC;            0594         LDI A.0(DELAY1)   ;PLO RC  ..INITIALIZE RC
8285 ;                  0595                                    ..TO POINT AT THE
8285 ;                  0596                                    ..DELAY ROUTINE
8285 F880BC;            0597         LDI A.1(DELAY1)   ;PHI RC
8288 F83FA5;            0598         LDI A.0(START)   ;PLO R5  ..INITIALIZE PC
828B F880B5;            0599         LDI A.1(START)   ;PHI R5
828E F800A2;            0600         LDI#00   ;PLO R2          ..R2 POINTS TO
8291 F88CB2;            0601         LDI A.1(WRAM)   ;PHI R2   ..M(8C00)
8294 6101;              0602         OUT 1,#01                 ..SELECT RCA I/O
8296 ;                  0603                                    ..GROUP
8296 7155;              0604         DIS,#55                   ..DISABLE INTRPT
8298 ;                  0605                                    ..P=X=5
8298 ;                  0606 ..  DSKGO ROUTINE
8298 ;                  0607 ..  THIS ROUTINE INITIALIZES R4,R5,RC TO
8298 ;                  0608 ..  POINT AT THE CALL, RET, AND DELAY
8298 ;                  0609 ..  ROUTINES RESPECTIVELY.
8298 ;                  0610 ..  THIS ROUTINE DYNAMICALLY DETERMINES THE
8298 ;                  0611 ..  STACK LOCATION,AND INITIALIZE R2 TO
8298 ;                  0612 ..  POINT AT THAT LOCATION.
8298 ;                  0613 ..  IT ALSO HOMES BOTH DSK DRIVES IF POSSIBLE.
8298 F805A3;            0614  DSKGO1:LDI#05   ;PLO R3
829B F800B3;            0615         LDI#00   ;PHI R3
829E ;                  0616  DSKGO2:ORG *
829E F864A4;            0617         LDI A.0(CALL)   ;PLO R4   ..INITIALIZE R4 TO
82A1 ;                  0618                                    ..POINT AT THE
82A1 ;                  0619                                    ..CALL ROUTINE
```

```
82A1 F874A5;          0620          LDI A.0(RET)    ;PLO R5   ..INITIALIZE R5 TO
82A4 ;                0621                                    ..POINT AT RET ROU-
82A4 ;                0622                                    ..TINE
82A4 F883B4;          0623          LDI A.1(CALL)   ;PHI R4   ..R4,R5 ON
82A7 B5;              0624          PHI R5                    ..SAME PAGE
82A8 F880B2;          0625          LDI#80    ;PHI R2
82AB F8FFA2;          0626          LDI#FF    ;PLO R2
82AE 92FF01B2;        0627  STACK:GHI R2  ;SMI#01   ;PHI R2 ..ASSUME 4K BANKS
82B2 ;                0628                                    ..OF MEMORY
82B2 F85A5202;        0629          LDI#5A    ;STR R2   ;LDN R2 ..CK IF RAM EXIST
82B6 FB5A3AAE;        0630          XRI#5A    ;BNZ STACK      ..BR IF NO RAM
82BA 650B;            0631  HOMDSK:OUT 5,#0B                   ..CLEAR ERROR FLAGS
82BC 6401;            0632          OUT 4,#01                 ..OUTPUT UNIT#00
82BE 6521;            0633          OUT 5,#21                 ..LOAD U/S#
82C0 E26E;            0634          SEX R2    ;INP 6          ..READ STATUS
82C2 FA20;            0635          ANI#20                    ..CK DRIVE
82C4 E0;              0636          SEX R0
82C5 3ACF;            0637          BNZ UNIT2                 ..BR IF DRIVE FAIL
82C7 650D;            0638          OUT 5,#0D                 ..SEEK TRACK#00
82C9 E26E;            0639  UNIT1:SEX R2    ;INP 6          ..READ STATUS
82CB F6;              0640          SHR                       ..CK FOR BUSY
82CC 33C9;            0641          BDF UNIT1                 ..BR IF BUSY
82CE E0;              0642          SEX R0
82CF 650B;            0643  UNIT2:OUT 5,#0B                   ..CLEAR ERROR FLAGS
82D1 6441;            0644          OUT 4,#41                 ..OUTPUT UNIT#1
82D3 6521;            0645          OUT 5,#21                 ..LOAD U/S#
82D5 E26E;            0646          SEX R2    ;INP 6          ..READ STATUS
82D7 FA20;            0647          ANI#20                    ..CK DRIVE
82D9 E0;              0648          SEX R0
82DA 3AE4;            0649          BNZ EXIT2                 ..BR IF DRIVE FAIL
82DC 650D;            0650          OUT 5,#0D                 ..SEEK TRACK#00
82DE E26E;            0651  UNIT:SEX R2    ;INP 6            ..READ STATUS
82E0 F6;              0652          SHR                       ..CK FOR BUSY
82E1 33DE;            0653          BDF UNIT                  ..BR IF BUSY
82E3 E0;              0654          SEX R0
82E4 6101;            0655  EXIT2:OUT 1,#01                   ..SELECT RCA I/O
82E6 ;                0656                                    ..GROUP
82E6 E2;              0657          SEX R2
82E7 D3;              0658          SEP R3
82E8 ;                0659 ..  THE FOLLOWING ROUTINE DOES (?R) COMMAND
82E8 ;                0660 ..
82E8 ;                0661 ..
82E8 ;                0662 ..
82E8 81;              0663 TYPER: GLO SWITCH                  ..CK IF ?
82E9 C280E1;          0664          LBZ SYNERR                ..BR IF NOT ?
82EC F89CA3;          0665          LDI A.0(TYPE5D) ;PLO SUB  ..SUB IS POINTING
82EF ;                0666                                    ..TO TYPE5D ROUTINE
82EF D30D;            0667          SEP SUB,#0D               ..TYPE CR
82F1 D30A;            0668          SEP SUB,#0A               ..TYPE LF
82F3 F802AD;          0669          LDI#02    ;PLO CNTER      ..TYPE R0,R1
82F6 D358;            0670 TYPEX:   SEP SUB,T'X'              ..TYPE X
82F8 D358;            0671          SEP SUB,T'X'              ..SINCE R0,R1 ARE
82FA D358;            0672          SEP SUB,T'X'              ..CLOBBERED BY UT20
82FC D358;            0673          SEP SUB,T'X'              ..X=DON'T CARE
82FE D320;            0674          SEP SUB,#20               ..TYPE SPACE
8300 2D;              0675          DEC CNTER
8301 8D;              0676          GLO CNTER
8302 CA82F6;          0677          LBNZ TYPEX                ..BR TO TYPE R1
8305 F804A0;          0678          LDI#04    ;PLO PTER       ..TYPE R2,R3
8308 F88CB0;          0679          LDI#8C    ;PHI PTER       ..LOAD ADDERSS
830B F802AD;          0680          LDI#02    ;PLO CNTER      ..LOAD CNTER
830E 40BF;            0681 TYPER2: LDA PTER ;PHI CHAR         ..PRINT 2 HEX DIGIT
8310 F8AEA3;          0682          LDI A.0(TYPE2) ;PLO SUB
```

```
8313 D3;          0683          SEP SUB
8314 40BF;        0684          LDA PTER   ;PHI CHAR
8316 F8AEA3;      0685          LDI A.0(TYPE2) ;PLO SUB
8319 D3;          0686          SEP SUB
831A F89CA3;      0687          LDI A.0(TYPE5D) ;PLO SUB
831D 80FB08;      0688          GLO PTER ;XRI#08
8320 C6;          0689          LSNZ
8321 D32C;        0690          SEP SUB,T',
8323 D320;        0691          SEP SUB,#20
8325 2D;          0692          DEC CNTER
8326 8D3A0E;      0693          GLO CNTER   ;BNZ TYPER2
8329 D358;        0694          SEP SUB,T'X'
832B D358;        0695          SEP SUB,T'X'
832D F809A0;      0696          LDI#09 ;PLO PTER
8330 F88CB0;      0697          LDI#8C ;PHI PTER
8333 F816AD;      0698          LDI#16 ;PLO CNTER
8336 40BF;        0699          LDA PTER ;PHI CHAR
8338 F8AEA3;      0700          LDI A.0(TYPE2) ;PLO SUB
833B D3;          0701          SEP SUB                    ..TYPE OTHER TWO
833C D320;        0702 TSPCE: SEP SUB; ,#20                ..SPACE
833E ;            0703 ..
833E 40BF;        0704 TLOOPX:  LDA PTER ;PHI CHAR         ..FETCH ONE BYTE
8340 ;            0705                                     ..FOR TYPING
8340 F8AEA3;      0706          LDI A.0(TYPE2) ;PLO SUB
8343 D3;          0707          SEP SUB                    ..TYPE 2 HEX
8344 2D;          0708          DEC CNTER
8345 8D;          0709          GLO CNTER
8346 3A4C;        0710          BNZ TL3A                   ..BRANCH NOT DONE
8348 9D;          0711          GHI CNTER
8349 C2803F;      0712          LBZ START                  ..BRANCH IF DONE
834C 80FB18;      0713 TL3A:    GLO PTER ;XRI#18           ..CK IF RC?
834F 3A53;        0714          BNZ TLX
8351 D32C;        0715          SEP SUB,T',
8353 80FA0F;      0716 TLX:     GLO PTER ;ANI #0F          ..PTER DIV BY 16?
8356 3A5E;        0717          BNZ TL2A
8358 D30D;        0718          SEP SUB; ,#0D              ..THEN CR
835A D30A;        0719          SEP SUB,#0A                ..TYPE LF
835C 303E;        0720          BR TLOOPX
835E F6;          0721 TL2A:     SHR                       ..DIV BY 2?
835F 333E;        0722          BDF TLOOPX                 ..NO, LOOP BACK
8361 303C;        0723          BR TSPCE                   ..ELSE TYPE SPACE &
8363 ;            0724                                     ..LOOP BACK
8363 ;            0725 ..  STANDARD CALL ROUTINE
8363 D3;          0726  EXITA:SEP R3                        ..R3 IS POINTING
8364 ;            0727                                     ..TO FIRST INSTR.
8364 ;            0728                                     ..IN SUBROUTINE
8364 E2;          0729  CALL:SEX R2                        ..POINT TO STACK
8365 96;          0730          GHI R6                     ..PUSH R6 ONTO
8366 73;          0731          STXD                       ..STACK TO PREPARE
8367 ;            0732                                     ..IT  FOR PONTING
8367 86;          0733          GLO R6                     ..TO ARGUMENTS,
8368 ;            0734                                     ..AND DECREMENT
8368 73;          0735          STXD                       ..TO FREE LOCATION.
8369 93;          0736          GHI R3                     ..COPY R3 INTO R6
836A B6;          0737          PHI R6                     ..TO SAVE RETURN
836B ;            0738                                     ..ADDRESS
836B 83;          0739          GLO R3                     ..SAVE THE RETURN
836C ;            0740                                     ..ADDRESS
836C A6;          0741          PLO R6                     ..SAVE THE RETURN
836D ;            0742                                     ..ADDRESS
836D 46;          0743          LDA R6                     ..LOAD THE ADDRESS
836E ;            0744                                     ..OF SUBROUTINE
836E B3;          0745          PHI R3                     ..INTO R3
```

```
836F 46;          0746          LDA R6                    ..INTO R3
8370 A3;          0747          PLO R3                    ..INTO R3
8371 3063;        0748          BR EXITA                  ..BRANCH TO ENTRY
8373 ;            0749                                    ..POINT
8373 ;            0750 ..  STANDARD RETURN ROUTINE
8373 D3;          0751  EXITR:SEP R3                      ..RETURN TO MAIN
8374 ;            0752                                    ..PROGRAM
8374 96;          0753  RET:GHI R6                        ..COPY R6 INTO R3
8375 B3;          0754          PHI R3                    ..R3 CONTAINS THE
8376 ;            0755                                    ..RETURN
8376 86;          0756          GLO R6                    ..ADDRESS
8377 A3;          0757          PLO R3                    ..ADDRESS
8378 E2;          0758          SEX R2                    ..POINT TO STACK
8379 12;          0759          INC R2                    ..POINT TO SAVED
837A ;            0760                                    ..OLD R6
837A 72;          0761          LDXA                      ..RESTORE THE
837B ;            0762                                    ..CONTENTS
837B A6;          0763          PLO R6                    ..OF R6
837C F0;          0764          LDX                       ..OF R6
837D B6;          0765          PHI R6                    ..OF R6
837E 9F;          0766          GHI RF
837F C08373;      0767         LBR EXITR                  ..BRANCH TO ENTRY
8382 ;            0768                                    ..POINT
8382 ;            0769 ..  UT20 VECTOR TABLE
8382 ;            0770          ORG#83F0
83F0 C0824D;      0771  OSTRNG:LBR MSGE
83F3 C08298;      0772  INIT1:LBR DSKG01
83F6 C0829E;      0773  INIT2:LBR DSKG02
83F9 C08281;      0774  GOUT20:LBR ENTER
83FC C0816C;      0775 CKHEX:  LBR CKHXE
83FF ;            0776 ..
83FF ;            0777 END
0000
```

```
!M
0000 ;            0001
0000 ;            0002
0000 ;            0003
0000 ;            0004
0000 ;            0005
0000 ;            0006          ORG #8400
8400 ;            0007         ..THIS ROUTINE IS USED TO LOAD A PROGRAM
8400 ;            0008         ..WRITTEN IN UT2 FORMAT FROM ICOM FDSK
8400 ;            0009         ..INTO MEMORY. THIS PROGRAM STARTS ASKING
8400 ;            0010         ..FOR THE TRACK# AND UNIT#.THESE
8400 ;            0011         ..NUMBERS SHOULD BE ENTERED FROM
8400 ;            0012         ..TERMINAL AS HEX DIGITS,THEN THE PROGRAM
8400 ;            0013         ..SEEKS THE U/TR AND LOAD THE PGM
8400 ;            0014         IRX=#60
8400 ;            0015         TYPE2=#81AE
8400 ;            0016         PTER=#0C          ..DCB(DATA CONTROL BLOCK) P
.0
```

```
8400 ;                0017          ST=#02              ..STACK POINTER
8400 ;                0018          PC=#03              ..MAIN PROGRAM COUNTER
8400 ;                0019          ASL=#0D
8400 ;                0020          AUX=#0E
8400 ;                0021          ..I/O PARAMETERS
8400 ;                0022         READAH=#813B
8400 ;                0023         TYPE=#81A4
8400 ;                0024         OSTRNG=#83F0
8400 ;                0025         DSKGO=#83F6
8400 ;                0026         GOUT20=#83F9
8400 ;                0027         CKHEX=#83FC
8400 F809A3;          0028         LDI A.0(START-#06)    ;PLO R3
8403 F884B3;          0029         LDI A.1(START)    ;PHI R3
8406 C083F6;          0030         LBR DSKGO
8409 F81FA2;          0031         LDI#1F    ;PLO R2
840C F88CB2;          0032         LDI#8C    ;PHI R2
840F F800ADBD;        0033  START: LDI#00    ;PLO ASL    ;PHI ASL      ..CLEAR ASL
8413 BA73;            0034         PHI RA    ;STXD                     ..CLEAR RA
8415 D483F00D0A5245;0035   ASK:    SEP R4 ,A(OSTRNG),#0D,#0A,T'READ?',#00
841C 41443F00;        0035
8420 D4813B;          0036   ASK1:  SEP R4; ,A(READAH)    ..READ A CHAR
8423 FB0D;            0037         XRI#0D                    ..CK FOR A CR
8425 3A20;            0038         BNZ ASK1
8427 D483F00A4C4F41;0039           SEP R4,A(OSTRNG),#0A,T'LOADING',#00
842E 44494E4700;      0039
8433 E3;              0040         SEX R3
8434 6400;            0041         OUT 4,#00             ..OUTPUT U/S#00
8436 9D;              0042         GHI ASL
8437 323D;            0043         BZ CONTIN             ..BRANCH IF U/S#00
8439 F840;            0044         LDI#40                ..UNIT#1
843B 6440;            0045         OUT 4,#40             ..OUTPUT U/S#40
843D 52;              0046 CONTIN: STR ST                ..STORE U/S# IN DCB
843E 226521;          0047         DEC ST    ;OUT 5,#21  ..LOAD THE U/S#
8441 650B;            0048         OUT 5,#0B             ..CLEAR ERROR FLAG
8443 650D;            0049         OUT 5,#0D             ..SEEK TRACK#00
8445 ;                0050         ..THE FOLLOWING ROUTINE CONVERTS A DECIMAL# I
.N R9.
8445 ;                0051         ..HEX AND STORE IT @DCB PTER
8445 E28D;            0052   CVY:    SEX R2;GLO ASL
8447 FF10;            0053          SMI#10
8449 3B52;            0054          BNF RESULT
844B AD;              0055          PLO ASL
844C 9A;              0056          GHI RA
844D FC0A;            0057          ADI#0A
844F BA;              0058          PHI RA
8450 3045;            0059          BR CVY    ..BRANCH IF NOT NEGATIVE
8452 8D;              0060   RESULT:GLO ASL
8453 52;              0061          STR ST
8454 9A;              0062          GHI RA
8455 F4;              0063          ADD
8456 73;              0064          STXD
8457 82AC;            0065          GLO ST    ;PLO PTER
8459 92BC1C;          0066          GHI ST     ;PHI PTER    ;INC PTER
845C 1C1C;            0067          INC PTER    ;INC PTER    ..PTER @THE BYTE COUN
.
845E D48506;          0068          SEP R4,A(EWAIT)  ..WAIT UNTIL DISK NOT BUSY
8461 D48573;          0069   READX: SEP R4; ,A(READ)    ..READ 1 ASCII DIGIT
8464 ;                0070                             ..FROM READ BUFFER->RF.1
8464 CB83F9;          0071          LBNF GOUT20       ..READ ERROR RESTART
8467 FB21;            0072          XRI#21    ..CK FOR !
8469 327A;            0073          BZ ISITM
846B FB05;            0074          XRI #05           ..CHECK FOR $
846D 32C6;            0075          BZ ISITU
```

```
846F FB37;           0076          XRI #37          ..CHECK FOR EOF(DC3)
8471 3A61;           0077          BNZ READX
8473 D483F000;       0078 DONE:    SEP R4 ,A(OSTRNG) ,#00    ..TYPE NULL MESSAGE
. RESET DELAY PTR
8477 C083F9;         0079          LBR GOUT20          ..TRANSFER CONTROL TO UT2
.0
847A D48573;         0080  ISITM:  SEP R4; ,A(READ)
847D CB83F9;         0081          LBNF GOUT20      ..READ ERROR RESTART
8480 FB4D;           0082          XRI#4D    ..CK FOR M
8482 3AE1;           0083          BNZ ERRORX     ..IF NOT M->ERROR
8484 D484F6;         0084   READX1:SEP R4; ,A(READHX)    ..READ 1 ASCII DIGIT
8487 ;               0085                                  ..AND CK IF HEX
8487 3399;           0086          BDF READX2    ..BR IF HEX
8489 FB2E;           0087          XRI#2E    ..CK IF "."
848B 3A84;           0088          BNZ READX1
848D D48573;         0089   READXA:SEP R4; ,A(READ)
8490 CB83F9;         0090          LBNF GOUT20      ..READ ERROR RESTART
8493 FB0D;           0091          XRI#0D
8495 3A8D;           0092          BNZ READXA
8497 3084;           0093          BR READX1
8499 D484F6;         0094   READX2:SEP R4; ,A(READHX)     ..READ 2ND ASCII DIGIT
.
849C ;               0095                                  ..AND CK IF HEX
849C 3399;           0096          BDF READX2     ..BR IF HEX
849E FB20;           0097          XRI#20      ..CK IF SPACE
84A0 3AE1;           0098          BNZ ERRORX       ..BR IF NOT SPACE
84A2 8DA8;           0099          GLO ASL    ;PLO R8      ..ADDRESS->R8
84A4 9DB8;           0100          GHI ASL   ;PHI R8
84A6 D484F6;         0101   READX3:SEP R4; ,A(READHX)     ..READ AN ASCII DIGIT
84A9 ;               0102                                  ..AND CK FOR HEX
84A9 3BBC;           0103          BNF READX4      ..BR IF NOT HEX
84AB D484F6;         0104   READX8:SEP R4; ,A(READHX)     ..READ THE 2ND ASCII D
.I
84AE ;               0105                                  ..AND CK IF HEX
84AE 3BE1;           0106          BNF ERRORX    ..BR IF NOT HEX->ERROR
84B0 8D58;           0107          GLO ASL    ;STR R8    ..STORE AT THE SPECIFIED
.
84B2 ;               0108                               ..ADDRESS
84B2 E8F3;           0109         SEX R8; XOR      ..DID IT WRITE CORRECTLY?
84B4 32B9;           0110         BZ WRTOK         ..YES
84B6 D48779;         0111         SEP R4,A(NOTRAM)          ..NO
84B9 18;             0112 WRTOK:   INC R8
84BA 30A6;           0113          BR READX3
84BC FB0D;           0114   READX4:XRI#0D    ..CK IF CR
84BE 3261;           0115          BZ READX     ..IF CR->DONE
84C0 FB36;           0116          XRI#36    ..CK FOR SEMICOLON
84C2 ;               0117                               ..TEST WITH(CR.XOR.,.XOR.
.;
84C2 328D;           0118          BZ READXA         ..BR IFSEMICOLON
84C4 30A6;           0119          BR READX3
84C6 D48573;         0120  ISITU:  SEP R4 ,A(READ)
84C9 CB83F9;         0121          LBNF GOUT20      ..READ ERROR RESTART
84CC FB55;           0122          XRI T'U'
84CE 3AE1;           0123          BNZ ERRORX
84D0 D484F6;         0124  ADLP:   SEP R4 ,A(READHX)
84D3 33D0;           0125          BDF ADLP
84D5 8DA0;           0126          GLO ASL    ;PLO R0
84D7 9DB0;           0127          GHI ASL    ;PHI R0
84D9 D483F00D0A00;   0128          SEP R4; ,A(OSTRNG),#0D0A,#00       ..OUTPUT A
. LF
84DF E0;             0129          SEX R0
84E0 D0;             0130          SEP R0
84E1 D483F00D0A464F;0131  ERRORX: SEP R4; ,A(OSTRNG),#0D0A,T'FORMAT ERROR',#00
.
84E8 524D4154204552;0131
84EF 524F5200;       0131
```

```
84F3 C083F9;          0132            LBR GOUT20
84F6 ;                0133            ..SUBROUTINES
84F6 ;                0134            ..THIS ROUTINE READS 1 ASCII DIGIT FROM DISK
.
84F6 D48573;          0135  READHX: SEP R4; ,A(READ)    ..READ 1 ASCII DIGIT
84F9 CB83F9;          0136           LBNF GOUT20       ..READ ERROR RESTART
84FC D483FC;          0137           SEP R4; ,A(CKHEX)    ..CK IF HEX
84FF D5;              0138  EXIT:    SEP R5
8500 ;                0139           ORG #8500
8500 ;                0140 ..
8500 ;                0141 ............BRANCH POINTS
8500 ;                0142 ..
8500 30C6;            0143 EWRITE: BR WRITE          ..ENTRY TO DISK WRITE ROUTINE
.
8502 3073;            0144 EREAD: BR READ  ..ENTRY TO DISK READ ROUTINE
8504 3017;            0145 ETRNFR: BR TRNFR1
8506 3011;            0146 EWAIT: BR WAIT1 ..ENTRY TO SIMPLE WAIT ROUTINE
8508 C08629;          0147 DER: LBR DERROR
850B C0860E;          0148 EWAITD: LBR WAIT
850E C087A0;          0149 LINEPR: LBR PRNTRF        ..LINE PRINTER UTILITY
8511 ;                0150 ..
8511 ;                0151 ............SUBROUTINE WAIT1
8511 ;                0152 ..
8511 E2;              0153 WAIT1:  SEX R2
8512 6E;              0154         INP 6   ..GET DISK STATUS
8513 F6;              0155         SHR     ..BUSY=>DF
8514 3311;            0156         BDF WAIT1
8516 D5;              0157         SEP R5  ..RETURN
8517 ;                0158 ..
8517 ;                0159 ............SUBROUTINE TRNFR1
8517 ;                0160 ..
8517 F810AF;          0161 TRNFR1: LDI #10; PLO RF    ..16 ERRORS ALLOWED
851A 4C52;            0162         LDA PTER; STR R2          ..GET TRK #,STORE ON
.STACK
851C ;                0163                                   ..POINT TO UNIT-SECT #
851C 6422;            0164         OUT 4; DEC R2    ..OUTPUT THE TRACK #
851E E36511E2;        0165         SEX R3; OUT 5 ,#11; SEX R2   ..LOAD TRK #
8522 4C52;            0166         LDA PTER; STR R2          ..GET UNIT-SECT #,STO
.RE ON STACK
8524 ;                0167                                   ..POINT TO BYTE COUNT
8524 6422;            0168         OUT 4; DEC R2    ..OUTPUT UNIT-SECT #
8526 E36521;          0169         SEX R3;OUT 5 ,#21    ..LOAD UNIT-SECT #
8529 C4C4C4C4;        0170         NOP; NOP; NOP; NOP        ..WAIT 48US FOR DISK
852D 6509;            0171         OUT 5 ,#09       ..SEEK TRACK
852F D4860E;          0172         SEP R4,A(WAIT)            ..WAIT TO SEEK
8532 9FFE;            0173         GHI RF;SHL       ..ERROR FLAG=>DF
8534 336E;            0174         BDF TRNEXT       ..DRIVE FAIL ERROR, RETURN
8536 6EFA08;          0175         INP 6; ANI #08   ..CHECK FOR CRC ERROR
8539 3241;            0176         BZ RDWTCK       ..NO CRC ERROR
853B 9FF940BF;        0177         GHI RF;ORI #40;PHI RF    ..SET SEEK ERROR FLAG
.
853F 306E;            0178         BR TRKNG         ..PRINT SEEK ERROR
8541 E3650B;          0179 RDWTCK: SEX R3; OUT 5 ,#0B       ..CLEAR ERROR FLAGS
8544 9F;              0180         GHI RF           ..CK READ/WRITE FLAG
8545 F6;              0181         SHR      ..FLAG INTO DF
8546 334C;            0182         BDF WRTCK        ..BR IF WRITE
8548 6503;            0183         OUT 5 ,#03       ..READ
854A 3054;            0184         BR STATUS  ..WAIT FOR  READ
854C 6505;            0185 WRTCK:  OUT 5, #05       ..WRITE
854E D4860E;          0186         SEP R4,A(WAIT)            ..WAIT TO DO THE WRIT
.E
8551 E36507;          0187         SEX R3; OUT 5 ,#07       ..READ CRC
8554 D4860E;          0188 STATUS: SEP R4,A(WAIT)  ..WAIT FOR COMMAND TO EXECUTE
.
8557 6E;              0189         INP 6
8558 FA08;            0190         ANI #08    ..BIT3=1=>CRC ERROR
```

```
855A 326E;            0191          BZ TRNEXT        ..NO CRC ERROR RETURN
855C 2F;              0192          DEC RF           ..DEC THE ALLOWED ERROR COUNT
.
855D 8F;              0193          GLO RF  ..ANY MORE ALLOWED?
855E 3A41;            0194          BNZ RDWTCK        ..BR IF YES
8560 D48629;          0195          SEP R4,A(DERROR)
8563 9FFA01F920BF;    0196          GHI RF;ANI #01;ORI #20;PHI RF    ..SET CRC FLA
.G
8569 306E;            0197          BR TRNEXT        ..RETURN
856B D48629;          0198 TRKNG:   SEP R4,A(DERROR)           ..PRINT ERROR
856E 9FFBFFFE;        0199 TRNEXT:  GHI RF;XRI #FF;SHL       ..SET/RESET ERROR FLA
.G=>DF
8572 D5;              0200          SEP R5 ..RETURN
8573 ;                0201 ..
8573 ;                0202 ............SUBROUTINE READ
8573 ;                0203 ..
8573 EC;              0204 READ:    SEX PTER
8574 F800BF;          0205          LDI #00;PHI RF  ..SET READ MODE
8577 F0;              0206          LDX      ..GET BYTE COUNT
8578 3AB0;            0207          BNZ SHFTBR    ..BUFFER NOT EMPTY, SHIFT BUFFER
.
857A F800BF;          0208 REREAD:  LDI #00;PHI RF  ..SET READ MODE
857D F88073;          0209          LDI #80; STXD   ..INITIALIZE BYTE COUNT FOR N
.EXT SECTOR
8580 ;                0210                ..POINT AT UNIT-SECT #
8580 F0FC0173;        0211          LDX; ADI #01; STXD   ..INCR SECTOR #, POINT A
.T TRK #
8584 FA1F;            0212          ANI #1F   ..MASK OUT UNITS BITS
8586 FD1A;            0213          SDI #1A   ..SECTOR > 26 ?
8588 3395;            0214          BDF CNTOK    ..NO, CHECK TRACK #
858A 1C;              0215          INC PTER   ..POINT AT UNIT SECT #
858B F0FAC0FC0173;    0216          LDX; ANI #C0; ADI #01; STXD  ..RESET SECT # T
.O 1, POINT TRK #
8591 F0FC015C;        0217          LDX; ADI #01; STR PTER  ..INCR TRK #
8595 F0;              0218 CNTOK:   LDX      ..GET TRK #
8596 FD4C;            0219          SDI #4C    ..TRK # > 76 ?
8598 33A1;            0220          BDF TRKOK    ..TRACK IN RANGE, OK
859A 1C1C;            0221          INC PTER; INC PTER       ..POINT BYTE COUNT
859C E36500;          0222 SEX R3;OUT 5,#00         ..SET UP TO READ STATUS
859F 306B;            0223          BR TRKNG         ..PRINT TRACK RANGE ERROR
85A1 E2;              0224 TRKOK:   SEX R2
85A2 D48517;          0225          SEP R4,A(TRNFR1)              ..READ A SECTOR
.FROM DISK TO BUFFER
85A5 3BBC;            0226          BNF RDXIT        ..ERROR NOT CRC
85A7 EC;              0227          SEX PTER
85A8 9FFA203A7A;      0228          GHI RF;ANI #20;BNZ REREAD        ..READ NEXT S
.ECTOR ON CRC ERR
85AD F840;            0229          LDI #40    ..EXAMINE READ BUFFER
85AF C8;              0230          LSKP
85B0 F841;            0231 SHFTBR:  LDI #41    ..SHIFT READ BUFFER
85B2 52;              0232          STR R2
85B3 E2;              0233          SEX R2
85B4 6522;            0234          OUT 5; DEC R2
85B6 6E;              0235          INP 6    ..READ A BYTE
85B7 AF;              0236          PLO RF        ..SAVE IT
85B8 0CFF015C;        0237          LDN PTER; SMI #01; STR PTER  ..DEC BYTE COUNT
.
85BC 9FFBFFFE;        0238 RDXIT:   GHI RF;XRI #FF;SHL       ..SET/RESET ERROR FLA
.G=>DF
85C0 8F;              0239          GLO RF     ..GET READ BYTE
85C1 CF;              0240          LSDF     ..IF NO ERROR RETURN CHARACTER
85C2 F813;            0241          LDI #13 ..IF ERROR RETURN DC3
85C4 BF;              0242          PHI RF
85C5 D5;              0243          SEP R5         ..RETURN
85C6 ;                0244 ..
```

```
85C6 ;                  0245 ............SUBROUTINE WRITE
85C6 ;                  0246 ..
85C6 9F52;              0247 WRITE:  GHI RF; STR R2  ..SAVE DATA BYTE TO STACK
85C8 6422;              0248         OUT 4; DEC R2   ..OUTPUT THE DATA
85CA F801BF;            0249         LDI #01;PHI RF  ..SET WRITE MODE
85CD E36531;            0250         SEX R3; OUT 5 ,#31        ..LOAD WRITE BUFFER
85D0 EC;                0251         SEX PTER          ..POINT TO BYTE COUNT
85D1 F0FC01;            0252         LDX; ADI #01    ..INC THE BYTE COUNT
85D4 5C;                0253         STR PTER
85D5 FF80;              0254         SMI #80           ..BYTE COUNT< 128 ?
85D7 CB8609;            0255         LBNF EXWT          ..BR IF YES
85DA F801BF;            0256 REWRIT: LDI #01;PHI RF  ..SET WRITE MODE
85DD F80073;            0257         LDI #00;STXD          ..ZERO THE BYTE COUNT
85E0 ;                  0258                           ..POINT AT THE SEC#
85E0 F0FC0173;          0259         LDX; ADI #01; STXD       ..INC SEC# AND POINT
.TRK#
85E4 FA1F;              0260         ANI #1F    ..MASK OUT UNIT NUMBER
85E6 FD1A;              0261         SDI #1A          ..SECTOR > 26 ?
85E8 33F5;              0262         BDF WTCNT        ..NO, CHECK TRK #
85EA 1C;                0263         INC PTER         ..POINT AT UNIT/SEC#
85EB F0FAC0FC0173;      0264         LDX; ANI #C0; ADI #01; STXD ..RESET SECT # TO
. 1, POINT TRK #
85F1 F0FC01;            0265         LDX; ADI #01    ..INC THE TRACK#
85F4 5C;                0266         STR PTER
85F5 F0;                0267 WTCNT:  LDX      .. GET THE TRK #
85F6 FD4C;              0268         SDI #4C    ..TRK # > 76 ?
85F8 33FE;              0269         BDF TRKOK1       ..TRACK IN RANGE, OK
85FA 1C1C;              0270         INC PTER;INC PTER        ..POINT TO BYTE COUNT
.
85FC 306E;              0271         BR TRKNG         ..PRINT TRACK RANGE ERROR
85FE E2;                0272 TRKOK1: SEX R2
85FF D48517;            0273         SEP R4,A(TRNFR1)         ..WRITE BUFFER TO DIS
.K
8602 EC;                0274         SEX PTER
8603 9FFA20CA85DA;      0275         GHI RF;ANI #20;LBNZ REWRIT        ..CRC ERROR
.WRITE NEXT SECTOR
8609 9FFBFFFE;          0276 EXWT:   GHI RF;XRI #FF;SHL       ..SET/RESET ERROR FLA
.G=>DF
860D D5;                0277         SEP R5  ..RETURN
860E ;                  0278 ..
860E ;                  0279 ..
860E ;                  0280 ............SUBROUTINE WAIT
860E ;                  0281 ..
860E E2;                0282 WAIT:   SEX R2
860F 6E;                0283         INP 6   ..GET STATUS
8610 FA20;              0284         ANI #20      ..DRIVE FAIL ?
8612 3A19;              0285         BNZ FAILUR       ..DRIVE FAILED, PRINT ERROR
8614 6E;                0286         INP 6   ..GET STATUS
8615 FA40;              0287         ANI #40 ..DRIVE ACTIVE?
8617 3A1F;              0288         BNZ NOFAIL       ..YES, NO FAILURE
8619 F800AF;            0289 FAILUR: LDI #00;PLO RF  ..CLEAR TRY COUNT, DRIVE FAIL
.
861C D48629;            0290         SEP R4,A(DERROR)         ..PRINT DRIVE FAILURE
.
861F 6E;                0291 NOFAIL: INP 6          ..GET STATUS
8620 FA08;              0292         ANI #08          ..CRC ERROR?
8622 3A28;              0293         BNZ RETWAT       ..IF ERROR RETURN
8624 6EF6;              0294         INP 6;SHR        ..CHECK IF OPERATION DONE
8626 330E;              0295         BDF WAIT         ..NOT DONE
8628 D5;                0296 RETWAT: SEP R5  ..RETURN
8629 ;                  0297 ..
8629 ;                  0298 ...........SUBROUTINE DISK ERROR
8629 ;                  0299 ..
8629 E2;                0300 DERROR: SEX R2
862A 9C738C73;          0301         GHI RC;STXD;GLO RC;STXD ..SAVE DCB POINTERS
```

```
862E 9F738F73;      0302          GHI RF;STXD GLO RF;STXD ..SAVE FLAGS AND ERRO
.R COUNT
8632 9A738A73;      0303          GHI RA;STXD;GLO RA;STXD ..SAVE RA
8636 2C;            0304          DEC RC  ..POINT TO UNIT-SECT#
8637 0CFA7FAA;      0305          LDN RC;ANI #7F;PLO RA   ..SAVE UNIT-SECTOR# I
.N RA.0
863B 2C;            0306          DEC RC  ..POINT TO TRK#
863C 0CBA;          0307          LDN RC;PHI RA   ..SAVE TRACK# IN RA.1
863E 9F73;          0308          GHI RF;STXD   ..SAVE FLAGS
8640 FA40;          0309          ANI #40         ..CHECK FOR SEEK ERROR
8642 325C;          0310          BZ ERR10        ..NOT SEEK ERROR
8644 D483F00D0A5452;0311          SEP R4,A(OSTRNG),#0D0A,T'TRACK SEEK ERROR',#0
.0
864B 41434E20534545;0311
8652 4E204552524F52;0311
8659 00;            0311
865A 30BE;          0312          BR RDWRPT
865C 6EFA20;        0313 ERR10:   INP 6;ANI #20   ..CHECK FOR DRIVE FAILURE
865F 3276;          0314          BZ ERR20        ..NOT DRIVE RFAILURE
8661 D483F00D0A4452;0315          SEP R4,A(OSTRNG),#0D0A,T'DRIVE FAILURE',#00
8668 49564520464149;0315
866F 4C55524500;    0315
8674 30BE;          0316          BR RDWRPT
8676 6EFA08;        0317 ERR20:   INP 6;ANI #08   ..CHECK FOR CRC ERROR
8679 3290;          0318          BZ ERR30        ..NOT CRC ERROR
867B D483F00D0A4352;0319          SEP R4,A(OSTRNG),#0D0A,T'CRC ERROR',#00
8682 43204552524F52;0319
8689 00;            0319
868A 8AF980AA;      0320          GLO RA;ORI #80;PLO RA   ..SET CRC FLAG
868E 30BE;          0321          BR RDWRPT
8690 6EFA40;        0322 ERR30:   INP 6;ANI #40   ..DRIVE ACTIVE?
8693 3AAF;          0323          BNZ ERR40       ..YES
8695 D483F00D0A4452;0324          SEP R4,A(OSTRNG),#0D0A,T'DRIVE NOT ACTIVE',#0
.0
869C 495645204E4F54;0324
86A3 20414354495645;0324
86AA 00;            0324
86AB 60;            0325          ,IRX
86AC C08738;        0326          LBR DERXT       ..EXIT
86AF D483F00D0A5452;0327 ERR40:   SEP R4,A(OSTRNG),#0D0A,T'TRACK >76',#00
86B6 41434E203E3736;0327
86BD 00;            0327
86BE 60F0;          0328 RDWRPT:  ,IRX;LDX  ..GET FLAGS
86C0 F6;            0329          SHR     ..WRITE FLAG=>DF
86C1 33D5;          0330          BDF WROP        ..PRINT "DURING WRITE"
86C3 D483F020445552;0331 RDOP:    SEP R4,A(OSTRNG),T' DURING READ',#00
86CA 494E4720524541;0331
86D1 4400;          0331
86D3 30E6;          0332          BR UTSPT        ..PRINT "UNIT TRACK SECTOR"
86D5 D483F020445552;0333 WROP:    SEP R4,A(OSTRNG),T' DURING WRITE',#00
86DC 494E4720575249;0333
86E3 544500;        0333
86E6 D483F020554E49;0334 UTSPT:   SEP R4,A(OSTRNG),T' UNIT ',#00
86ED 542000;        0334
86F0 8AFA40;        0335          GLO RA;ANI #40  ..GET UNIT-SECT # AND TEST UN
.IT BIT
86F3 CE;            0336          LSZ     ..NOT UNIT 1
86F4 F801;          0337          LDI #01
86F6 FC30;          0338          ADI #30
86F8 BF;            0339          PHI RF
86F9 D481A4;        0340          SEP R4,A(TYPE)  ..TYPE UNIT #
86FC D483F02C205452;0341          SEP R4,A(OSTRNG),T', TRACK ',#00
8703 41434E2000;    0341
```

```
8708 9ABF;           0342          GHI RA; PHI RF  ..GET TRACK #
870A D4874D;         0343          SEP R4,A(TYPBCD)          ..TYPE TRACK #
870D D483F02C205345;0344           SEP R4,A(OSTRNG),T', SECTOR ',#00
8714 43544F522000;   0344
871A 8AFA1FBF;       0345          GLO RA;ANI #1F;PHI RF    ..GET SECTOR #
871E D4874D;         0346          SEP R4,A(TYPBCD)          ..TYPE SECTOR #
8721 8AFA803232;     0347          GLO RA;ANI #80;BZ NOSKIP          ..CHECK CRC F
.LAG
8726 D483F020534E49;0348           SEP R4,A(OSTRNG),T' SKIPPED',#00
872D 5050454400;     0348
8732 D483F00D0A00;   0349 NOSKIP: SEP R4,A(OSTRNG),#0D0A00          ..TYPE CR-LF
8738 60;             0350 DERXT:   ,IRX      ..RECOVER REGISTERS FROM STACK
8739 72AA72EA;       0351          LDXA;PLO RA;LDXA;PHI RA ..GET OLD RA
873D 72AF72;         0352          LDXA;PLO RF;LDXA      ..GET FLAGS AND ERROR CO
.UNT
8740 FA01F980BF;     0353          ANI #01;ORI #80;PHI RF  ..SET ERROR FLAG
8745 72ACF0BC;       0354          LDXA;PLO RC;LDX;PHI RC  ..GET DCB POINTER
8749 E3650B;         0355          SEX R3;OUT 5,#0B        ..CLEAR ERROR FLAGS
874C D5;             0356          SEP R5  ..RETURN
874D ;               0357 ..
874D ;               0358 ............SUBROUTINE TO PRINT HEX IN RF.1 AS BCD #
874D ;               0359 ..
874D 9F;             0360 TYPBCD: GHI RF  ..GET INPUT
874E FAF0;           0361          ANI #F0 ..STORE AS 2 HEX DIGITS
8750 F6F6F6F6;       0362          SHR;SHR;SHR;SHR
8754 AF;             0363          PLO RF
8755 9FFA0FBF;       0364          GHI RF;ANI #0F;PHI RF
8759 FF0A;           0365          SMI 10  ..DECIMAL ADJUST LOW DIGIT
875B 3B61;           0366          BNF SIXLP
875D 9FFC06BF;       0367          GHI RF;ADI 6;PHI RF
8761 8F;             0368 SIXLP:   GLO RF  ..ADD 16 TO BCD NUMBER FOR EACH HIGH
.HEX COUNT
8762 3275;           0369          BZ EXITBC       ..IF HIGH COUNT=0, EXIT
8764 2F;             0370          DEC RF
8765 9FFC16BF;       0371          GHI RF;ADI #16;PHI RF
8769 FA0F;           0372          ANI #0F
876B FF0A;           0373          SMI 10  ..DECIMAL ADJUST BCD RESULT
876D 3B61;           0374          BNF SIXLP
876F 9FFC06BF;       0375          GHI RF;ADI 6;PHI RF
8773 3061;           0376          BR SIXLP        ..LOOP UNTIL DONE
8775 D481AE;         0377 EXITBC: SEP R4,A(TYPE2)
8778 D5;             0378          SEP R5  ..RETURN
8779 ;               0379 ..................................................
8779 8C73;           0380 NOTRAM: GLO RC;STXD
877B 9C73;           0381          GHI RC;STXD
877D D483F00D0A5241;0382           SEP R4,A(OSTRNG),#0D0A,T'RAM AT ',#00
8784 4D2041542000;   0382
878A 98BF;           0383          GHI R8;PHI RF
878C D481AE;         0384          SEP R4,A(TYPE2)
878F 88BF;           0385          GLO R8;PHI RF
8791 D481AE;         0386          SEP R4,A(TYPE2)
8794 D483F0203F00;   0387          SEP R4,A(OSTRNG),T' ?',#00
879A 12;             0388          INC R2
879B 42BC;           0389          LDA R2;PHI RC
879D 02AC;           0390          LDN R2;PLO RC
879F D5;             0391          SEP R5
87A0 ;               0392 ..................................................
87A0 ;               0393 ..THIS ROUTINE PRINTS TO THE LINE PRINTER, THE CONTEN
.TS OF RF.1.
87A0 ;               0394 ..IT SUPRESSES PRINTING OF LINE FEEDS, AND REPLACES C
.ARRIAGE RETURNS
87A0 ;               0395 ..WITH A CR-LF PAIR.
```

```
87A0 ;                  0396 ..NORMALLY, THIS ROUTINE RETURNS WITH THE DFLAG SET,
.BUT IF THE
87A0 ;                  0397 ..CHARACTER IN RF.1 WAS A DC3(END OF FILE), THE DFLAG
. WILL BE RESET
87A0 ;                  0398 ..ON RETURN.
87A0 ;                  0399 ..
87A0 ;                  0400 ..
87A0 9FFB0A;            0401 PRNTRF: GHI RF;XRI #0A  ..IF LINE FEED, EXIT
87A3 32BC;              0402         BZ EXITDF
87A5 9FFB13;            0403         GHI RF;XRI #13  ..IF DC3, EXIT
87A8 32C0;              0404         BZ EXITEF
87AA 9FFBFF52;          0405 PRINT1: GHI RF;XRI #FF;STR R2   ..INVERT DATA
87AE 34AE;              0406         B1 *                ..WAIT UNTIL READY
87B0 6622;              0407         OUT 6; DEC R2       ..OUTPUT CHARACTER
87B2 9FFB0D;            0408         GHI RF;XRI #0D      ..CARRIAGE RETURN?
87B5 3ABC;              0409         BNZ EXITDF          ..NO, EXIT
87B7 F80ABF;            0410         LDI #0A;PHI RF      ..YES, PRINT A LINE FEED
87BA 30AA;              0411         BR PRINT1
87BC F801F6;            0412 EXITDF: LDI #01;SHR         ..SET DFLAG
87BF D5;                0413         SEP R5              ..AND RETURN
87C0 F6;                0414 EXITEF: SHR                 ..RESET DFLAG
87C1 D5;                0415         SEP R5              ..AND RETURN
87C2 ;                  0416         END
0000
```

Appendix H - System Checkout Game - DEDUCE

For checking out the CDS and to help the user familiarize himself with its operation, a game program is provided on Utility Program UT20. The object of the game, called "Deduce", is to guess a four-digit number (all digits different selected by the CDS at random and stored in its memory. The player should try to guess the number in a minimum number of tries. He is allowed fifteen tries. After each guess, CDS tells how many of the four digits are correctly placed and how many are in the set but incorrectly placed.

The paper tape version provides two options. Version A exercises the deductive skills of the player. Version B exercises both his deductive skills and memory by always erasing the previously printed guess.

How to Play Deduce (Paper Tape Program)

1. Enter the program from the paper tape provided into memory as described in the first Section of this Manual.
2. Type $U1 to start the program. CDS gives the prompt message:

VERSION A or B = >

3. Player types either A or B (as he chooses).
4. a. If player chooses A, CDS responds by printing:

TURN XX
GUESS = >,

b. If player decides upon Version B, CDS prints:

TIME = ,

to which the player should respond with any number between 1 and 9 (time parameter). Afterwards, game continues as in step 4a:

TURN XX
GUESS= >

5. At this point the player types in his guess number:

XXXX

(Numbers are assumed to be different, however, it may be good strategy to make some, or all, equal; player should remember his is given up to 15 trials.)

Immediately after this, CDS tells him how many digits are correctly placed, and also how many belong to the set but are incorrectly placed:

RP=X (right place)
WP=X (wrong place)

If his guess is entirely correct, CDS types:
WIN= >XXXX (correct answer)

If his guess has reached the last of his possible trials, the CDS answer is:
LOSE= >XXXX (correct answer)
In both cases return is made to a new game.

6. a. If none of the above (WIN-LOSE) occurs, and player had chosen Version A, then CDS prints:

TURN XX
GUESS= >

and game continues as in step # 5, or

b. For player using Version B, CDS waits for a period of $\simeq$ 2X (time parameter) seconds before overwriting the guess just entered so as to make it unreadable.

After that it writes:

TURN XX
GUESS= >

and game continues as in step #5.

Re-start Game

A new game can be started at any time by typing in R instead of a digit at guessing time. CDS responds by printing:

LOSE= >XXXX correct answer)
VERSION A or B= > (new game)

How to Play Deduce (Magnetic Tape Program)

1. Enter the program from the magnetic tape provided into memory as described in the first Section of this Manual.
2. Type $U1 to start the program. CDS will prompt:
CRT?
to which the player should respond by typing "N".
This will tell the program that a CRT is not being used for display.
3. Next CDS will display the following self-explanatory message:

"This RCA Microprocessor is programmed to play a number guessing game. Your objective is to guess the four different hidden digits. The COSMAC will tell how many digits are in the proper place, and how many are misplaced. To play enter your guess."

TURN GUESS PROPER PLACE MISPLACED WRONG DIGITS
15 -

4. Now the player can enter his four-digit guess which

will be automatically displayed under the heading: GUESS.
Immediately CDS will respond by filling in the values of the columns corresponding to:

PROPER PLACE, MISPLACED, and WRONG DIGITS

If the guess is entirely correct the words YOU WIN! are displayed. If the player has reached the last of his 15 possible trials, the message YOU LOSE! is displayed along with the correct answer.

5. If neither of the above occurs (WIN/LOSE), the turn number is decremented by one, and the player can enter a new guess.
6. If the player is very anxious to know the right guess, he can Resign the game by typing R, in lieu of any of his guess digits. Again, in addition to the correct answer he will see the words: YOU LOSE!
7. In either case of winning or losing, a new game can be started by hitting CR.

Appendix I -
Conversion to Different Operating Voltages

The CDS as supplied is wired for 115-volt ac operation. Other voltages including 100, 220, 230, or 240 volts can be accomodated by a simple one-wire change on the back of the CDS. Locate the terminal block adjacent to the power supply. Move the top wire to one of the other tapes as required. The bottom three wires should not be moved.

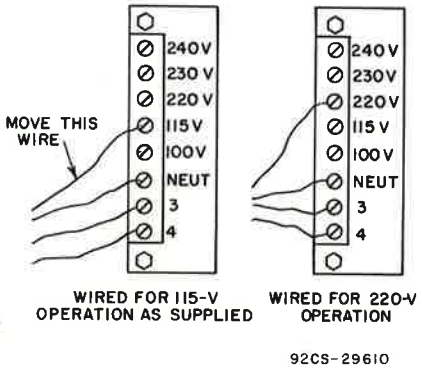


Fig. I1 — Conversion for different operating voltages.

Appendix J - Connection List for Terminal Interface Cables

TELETYPE TERMINAL

P1	P2	Signal
8	6	Data from TTY (Current Source)
7	8	Data to TTY (Current Source)
3	7	Data to TTY (Current Return)
4	5	Data from TTY (Current Return)
10	15	$+V_{DD}$
2	13	Paper Tape Control

EIA RS232C TERMINAL

P1	P2		
1	1	Ground	
2	2	Data to CDS	
3	3	Data to Terminal	
10	7	Signal Ground	
6	6 and 5	DSR	Held high by CDS
7	8	DCD	