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

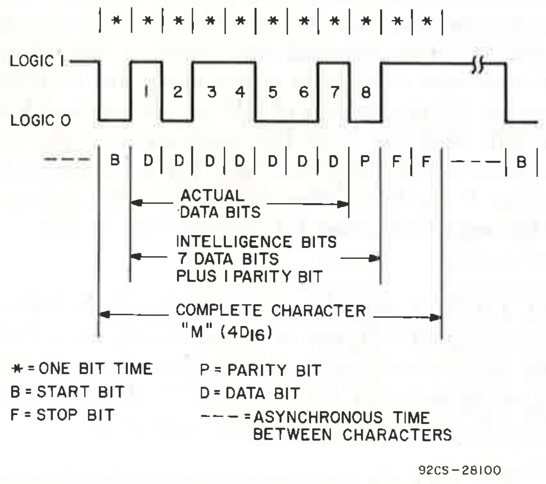


Fig. 2 — Data terminal bit serial output for the character "M".

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.

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

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

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

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 require's 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).

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.

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

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

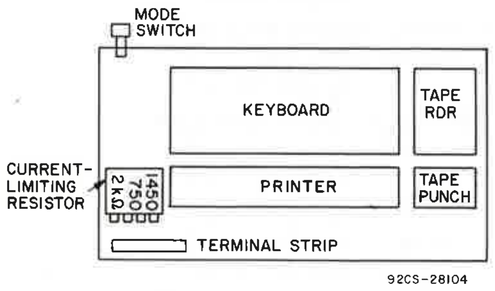
For a Teletype terminal connected for half-duplex operation, the following modifications can be made to convert it to full-duplex operation.

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.



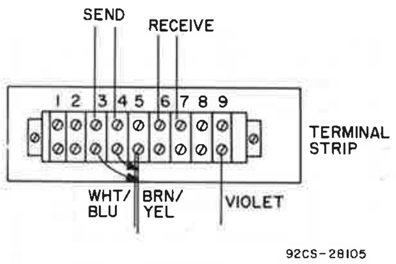


Fig. B1 — Location of and connections to terminal strip for Model 33 Teletype data terminal showing connections for 20-mA full-duplex operation.

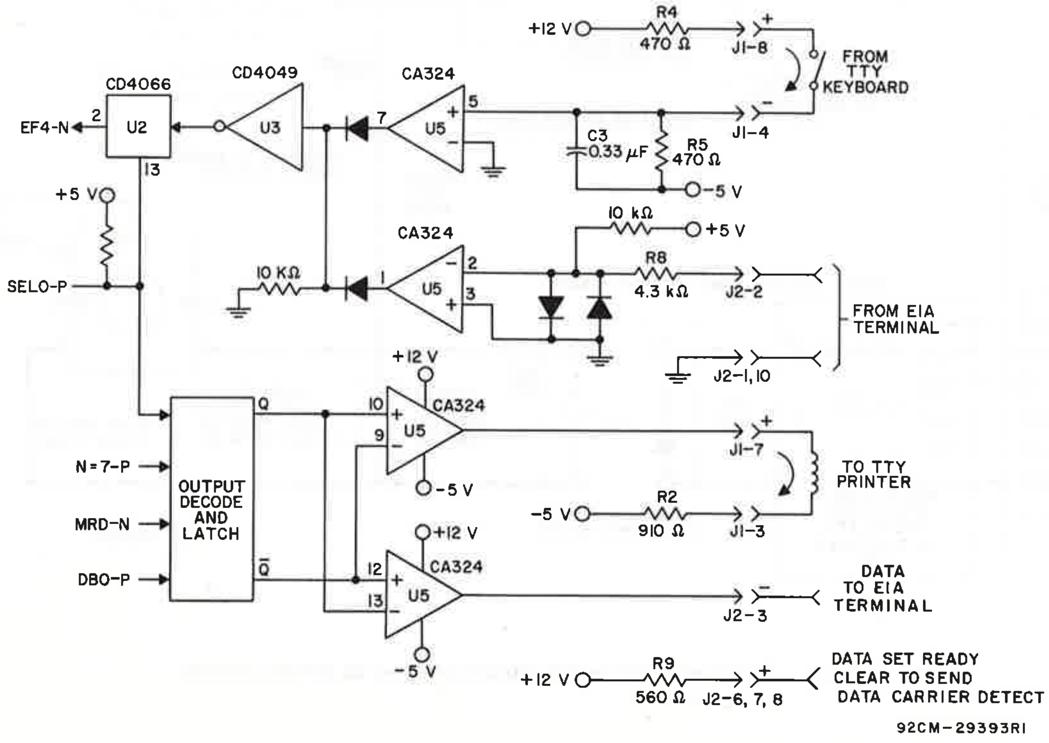


Fig. B2 — Detail of CPU-Terminal Interface (See Appendix D).