

August 1977

Using the Intel 8085 Serial I/O Lines

John Wharton
Microcomputer Applications

Using the Intel 8085 Serial I/O Lines

Contents

INTRODUCTION	1
MCS Family Members	1
Additional 8085 Instructions	1
CRT INTERFACE	3
Hardware Interface	3
Software Package	3
OUTPUT ROUTINE	5
INPUT ROUTINE	6
TIMING ANALYSIS	6
BAUD RATE IDENTIFICATION ROUTINE	7
CASSETTE RECORDER INTERFACE	8
Hardware Design	8
Software	9
OUTPUT ROUTINE	10
INPUT ROUTINE	10
ADDITIONAL COMMENTS	12
APPENDIX	13

INTRODUCTION

This application note is intended to acquaint the reader with the Intel® MCS-85 family, and to explain how to use one of its key features, a direct serial data link between the CPU and the outside world. Two design examples will be provided: a versatile method for direct communications between the CPU and a CRT or other peripheral at any rate from 110 to 9600 baud, and a magnetic tape interface system which allows programs and data to be stored or loaded using a cheap audio cassette recorder. Both examples use software routines to replace extensive external hardware and to provide additional flexibility.

MCS Family Members

The MCS-85 family consists of the new 8085 N-channel, 8-bit microprocessor (Figure 1) and a variety of parts which provide memory, input/output, timing, and peripheral control capability.

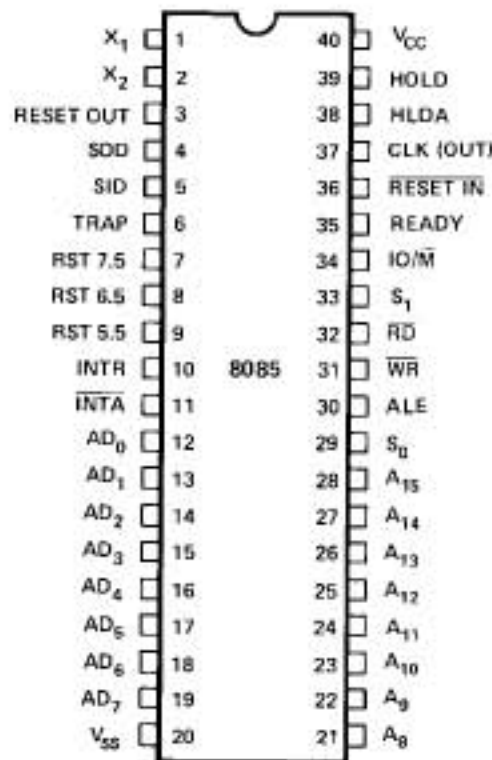


Figure 1. 8085 Pinout Diagram

In many respects the 8085 can be thought of as simply a hardware refinement of the extremely popular Intel® 8080 processor, which was introduced in 1973. It is 100% software-compatible with its predecessor. In addition to being 50% faster, many of the external timing and control functions needed by the 8080A have been integrated into the 8085 (such as the 8224 Clock

Generator and the 8228 System Controller), thus reducing the system part count. Additional features include four additional maskable and non-maskable interrupt pins, increased drive capability, and two new input and output lines. These pins, called SID and SOD, provide a serial I/O data link with the CPU. The 8085 uses a standard 40-pin DIP package.

All members of the MCS-85 family require a single 5-volt power supply, greatly reducing system overhead. Several components combine a number of system functions (e.g., ROM and I/O), allowing a complete, useful microcomputer system to be assembled with as few as three integrated circuits, as shown in Figure 2. For example, the 8155 and 8156 RAM/IO/TIMER chips each contain 256 8-bit bytes of program or data storage, three programmable I/O ports, and a 14-bit timer/counter.

Since the internal architecture and instruction set of the 8085 is an extension of that of the 8080, all software written for the 8080 – including compilers, assemblers, and individual applications programs – will run without modification on the new processor. In fact, the complete upward compatibility of the MCS-85 system means that applications designed around the 8080 can be converted to using the 8085 at minimal cost, requiring little hardware redesign or program modification. An engineer already familiar with the 8080 will not need to learn a new architecture or mnemonics. Companies now using Intel's extensive line of design, development, and debugging tools can augment their systems with a series of 8085 support products (such as the SDK-85, ICE-85, and SBC boards) which are compatible with their present mainframe and peripherals.

Additional 8085 Instructions

The additional hardware features of the 8085 (handling multiple-level maskable interrupts and serial I/O) are supported with two new instructions. RIM (machine code 20H) is used to read the current status of the three interrupt masks into the accumulator. Additional bits are set to show what interrupts (if any) are pending, and the logical state of the SID input pin (pin 5). The complement of RIM is SIM (machine code 30H), which has a dual function depending on the current accumulator contents. If bits 3 or 4 of the accumulator are a logical one, SIM can be used to change the three

interrupt masks; if bit 6=1, SIM can set the SOD output (pin 4). The two functions of the SIM instruction operate independently. (If, at this point, the acronyms RIM, SIM, SID, and SOD are starting to blur in your mind, try to remember their roots instead: Read Interrupt Mask, Set Interrupt Mask, Serial Input Data, and Serial Output Data. Don't worry; of the other four ordered permutations of R&S, I&O, and M&D, only ROM is used elsewhere in this note, and then only in its traditional sense.)

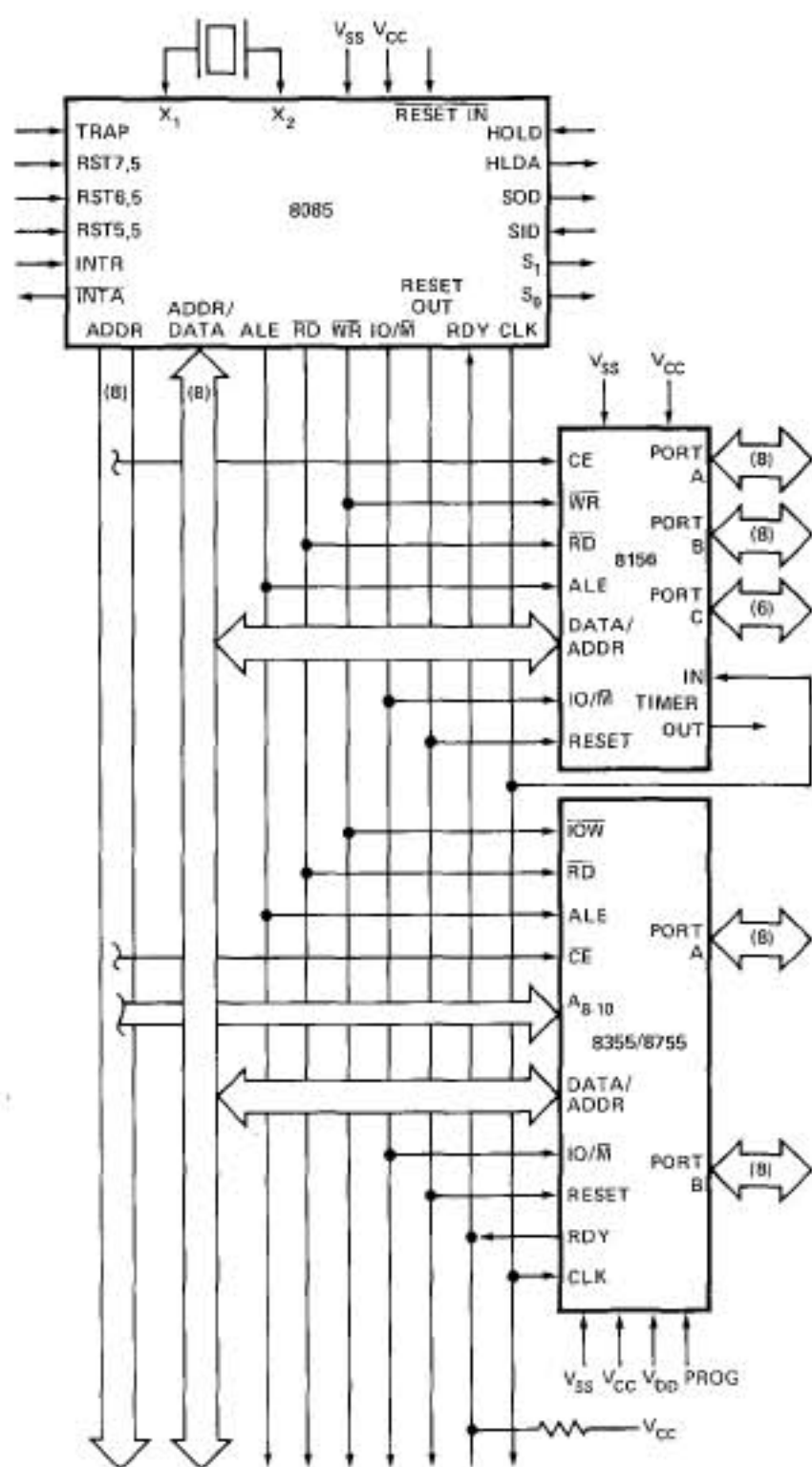


Figure 2. 8085 Minimum System

A detailed explanation of the accumulator contents after and before RIM and SIM is given in Figure 3. The I/O pins both function with respect to positive logic: a "1" in the accumulator corresponds to a high voltage level, "0" to a level near ground. The SID and SOD lines are electrically compatible with normal TTL logic levels. (For full electrical specifications, the reader should consult the MCS-85 User's Manual.) If A₆ is "0" prior to executing SIM, SOD will remain unchanged, regardless of the state of A₇. After a Reset, SOD will be low. It should be noted that RIM does not affect the Sign Flag; in order to make a conditional jump based on the Serial Input Data state, a three instruction sequence should be used, such as shown in Examples 1 and 2. When serial data is to be assembled into a parallel word, a sequence such as in Example 3 can be used, which shifts the contents of register pair HL one bit to the left and appends the input data bit.

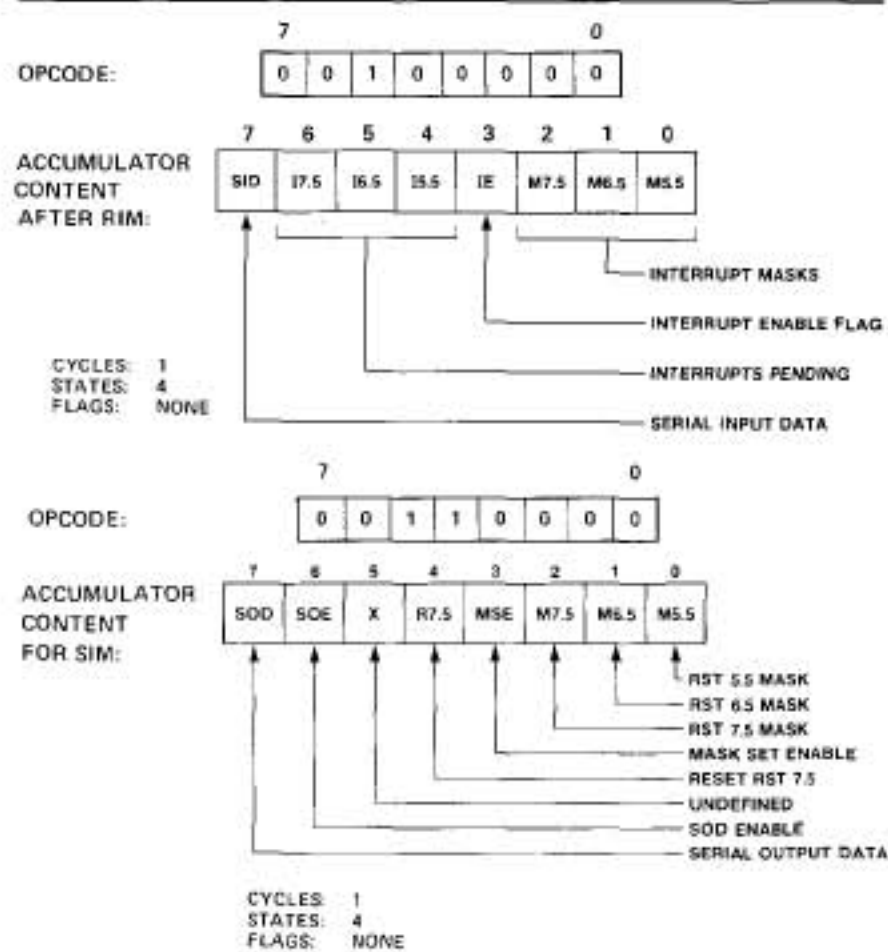


Figure 3. Effect of RIM and SIM Instructions

This note does not concern itself with the interrupt mask manipulation also made possible with RIM and SIM; for a full understanding of the interrupt capabilities of the 8085, see the User's Manual. To use SIM for altering the SOD state without fear of interfering with the interrupt mask status, one need only make sure that bits 3 and 4 of the accumulator are set to zero first.

EXAMPLE 1:

```

XXX
RIM          : READ SID LEVEL
ORR   A      : SET SIGN FLAG IF A7=1
JN    LABEL  : JUMP IF SID WAS HIGH.
XXX          : \ ELSE CONTINUE.

```

EXAMPLE 2:

```

XXX
RIM          : READ SID
RAL          : MOVE A7 INTO CY
CNC   SERVICE : CALL SERVICE ROUTINE
          : \ IF SID WAS LOW.
XXX          : \ THEN CONTINUE.

```

EXAMPLE 3:

```

XXX
RIM          : READ SID DATA BIT
RAL          : MOVE DATA INTO CY
MOV   A, L   :
RAL          : ROTATE DATA INTO L
MOV   L, A   :
MOV   A, H   :
RAL          : ROTATE OVERFLOW
MOV   H, A   : \ INTO H
XXX          : CONTINUE

```

On the following pages, examples are given showing two possible uses of the SID and SOD lines. The main purpose of these examples is not to give the reader a design after which he could model his own system — though, of course, this might be the case — but rather to illustrate the hardware and software interfaces and techniques necessary to implement a typical working subsystem.

CRT INTERFACE

Most microprocessor systems require some sort of serial communications. This may be selected for reasons of economy (to reduce the number of interconnections required in a distributed system), or it may be necessary in order to communicate with such common peripherals as CRT's or teletypewriters.

These peripherals all use a standard convention for transmitting serial ASCII code. Each data byte is transmitted as a series of 10 or 11 bits. The uniform time per bit corresponds to the data transmission rate. For example, if the transmission rate is to be 2400 baud (2400 bits per second), each bit time must be $1/2400 \text{ bps} = 416.7 \mu\text{sec/bit}$. The standard 10-bit sequence consists of a logically

zero "Start" bit, 8 data bits (least significant bit first), and one or more stop bits (logic 1). An 11-bit sequence with two stop bits is used for 110 baud TTY's. The logic one level continues until the start bit of the next byte to ensure that each 10-bit sequence is initiated with a one-to-zero transition. The 8 bits transferred might be raw binary data or alphanumeric characters using the standard ASCII code. In this case, the most significant bit — the last data bit transmitted — will depend on the parity convention being used. This sequence is illustrated for the ASCII "space" character in Figure 4.

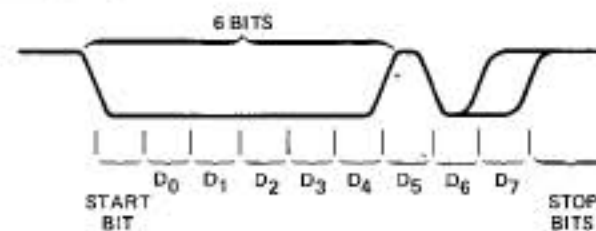


Figure 4. ASCII Space Character

The algorithm for receiving serial code involves sampling the incoming data at the middle of each bit time. The eight sampled values are shifted into a serial byte corresponding to the data originally transmitted. The one-to-zero transition at the beginning of each byte makes it possible to synchronize the sampling points relative to the start of each data sequence.

Hardware Interface

In general, any serial communications system will require both hardware and software interfaces. Since the SOD line can drive only one TTL load, additional current and voltage buffering is required to be compatible with the RS-232C interface standard used by most peripherals. A schematic for achieving this buffering is shown in Figure 5. The MC1488 and MC1489 circuits interface positive logic TTL signals with the RS-232 high voltage inverted logic levels.

Software Package

The software needed to drive the CRT interface is divided into three parts. All three use software timing and delay loops, with fixed and variable parameters. In conjunction, they are able to identify incoming signals at any rate from below 110 to over 9600 baud and respond at the same rate.

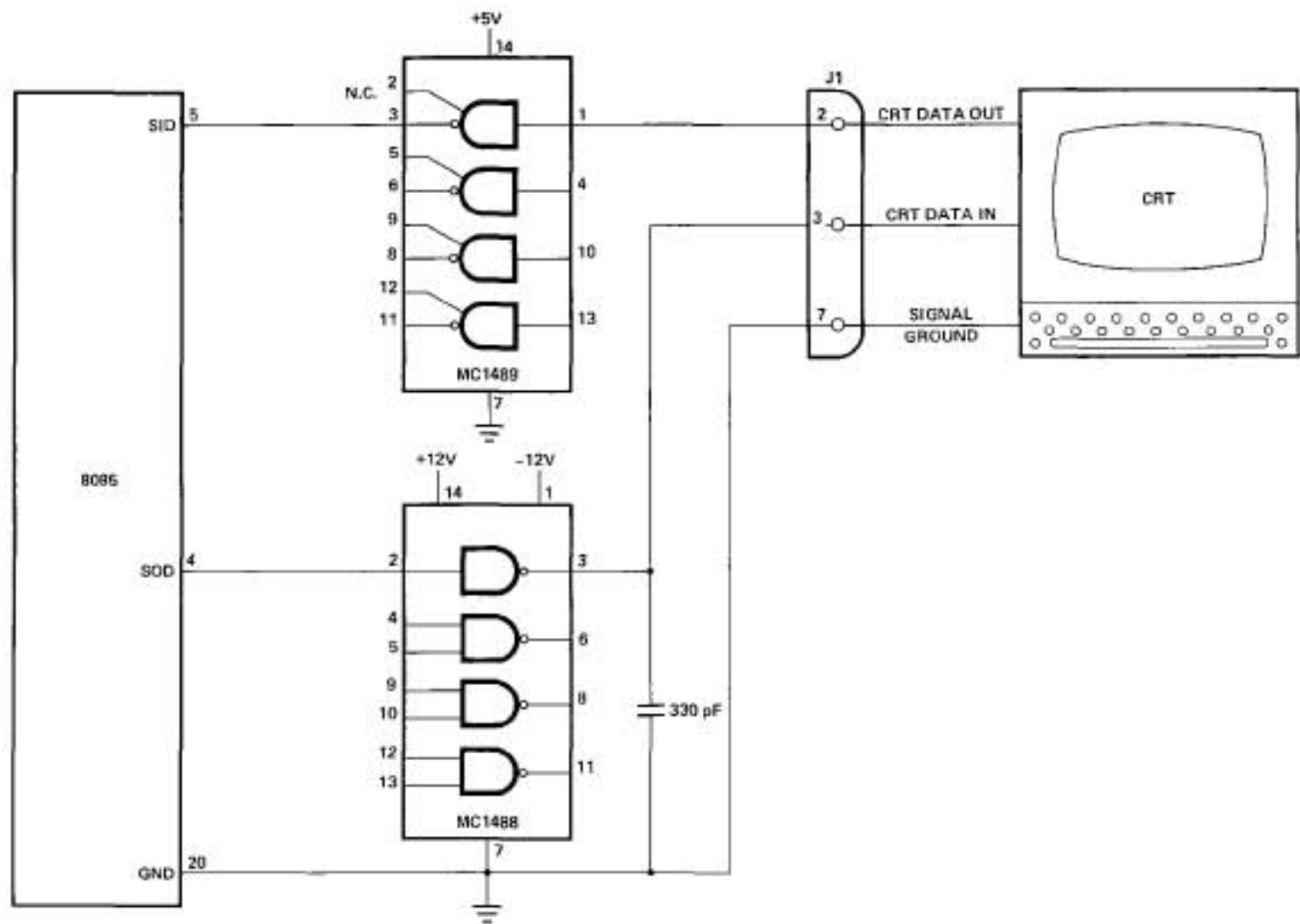


Figure 5. RS-232C Interface Schematic

Upon power-up or reset, or when the console device baud rate is changed, the baud rate identification subroutine (BRID) is called. This routine waits until an ASCII space character (20H) is received from the console. (Any other character will result in a case of mistaken identification.) When a space character is received, two time parameters are computed which correspond to the bit time and one-half the bit time of the baud rate being used. These are stored as variables BITTIME and HALFBIT. To output a character to the console, the character code is placed in register C, and the subroutine COUT is called. This routine uses BITTIME as a parameter for the software delay loop which determines the baud rate. To accept a character from the keyboard, CIN is called. CIN returns after the next key is typed, with the corresponding character code in register C. CIN uses both parameters BITTIME and HALFBIT.

Since COUT and CIN use time parameters computed by BRID, they will function at a rate the same as that of the initial space character input. Because of the nature of the software, the rate does not depend on the CPU clock frequency. This

results in additional flexibility in the following respects:

1. The software does not need to be modified if the 8085 crystal frequency is changed or Wait states are added.
2. Since the time base is no longer critical, the quartz crystal could be replaced by a less expensive RC network, provided the frequency does not drift by more than a few percent during a session. Additional drift can be accommodated by periodically recalling the BRID routine.
3. Communication is possible at non-standard baud rates which relaxes the constraints on system peripherals.

It should be noted, though, that slowing down the CPU clock will decrease its throughput proportionately. In addition, it will degrade the maximum resolution of the delay loops, with the result that the highest baud rates may no longer be achievable.

A more detailed analysis of the CRT interface routines will be presented in the order of increasing complexity: COUT, CIN, and BRID. Since SID and

SOD are ideal for many applications which involve critical I/O timing, the timing techniques used here may be of interest to software designers. Accordingly, the mathematical derivation of the timing parameters is included in this analysis, as well as a justification for the BRID algorithm. The algebra involved might be a bit too tedious for designers unconcerned with generating software delays. If so, they (and other bored readers) have the freedom of choice to skip over the sections they find objectionable.

OUTPUT ROUTINE

It would seem natural to write data in the standard format in three stages: output a zero start bit, then the 8 data bits (using a loop sequence), then the stop bits. Each stage would incorporate its own appropriate delay and output sections, leading to unnecessary duplication. Instead, the code below executes the same main loop 11 times. Its bit manipulation routine inherently results in the correct data sequence being formed. It accomplishes this by using the carry and C register as a 9-bit pseudo-circular shift register. Initially CY=0. The algorithm outputs CY, waits one bit time, sets CY=1, and then rotates the pseudo-register right one bit. This repeats for 11 cycles. On the tenth and all subsequent loops, the output bit will be a logical one, since that bit had been set nine loops earlier while in the CY (see Figure 6).

When COUT is called the registers to be used must be preserved and interrupts disabled so the timing loop will not be disrupted. Clear the CY in preparation for outputting the start bit, and set the loop counter for 11 bits (if 110 baud will never be used, the counter could be set to 10):

```
COUT:  PUSH  B
        PUSH  H
        DI
        XRA  A
        MVI  B, 11
```

Output of the contents of the CY:

```
CO1:   MVI  A, 80H    <7>
        RAR          <4>
        SIM          <4>
```

The numbers in brackets indicate how many machine cycles are required for each instruction. They will be referred to in the timing analysis section.

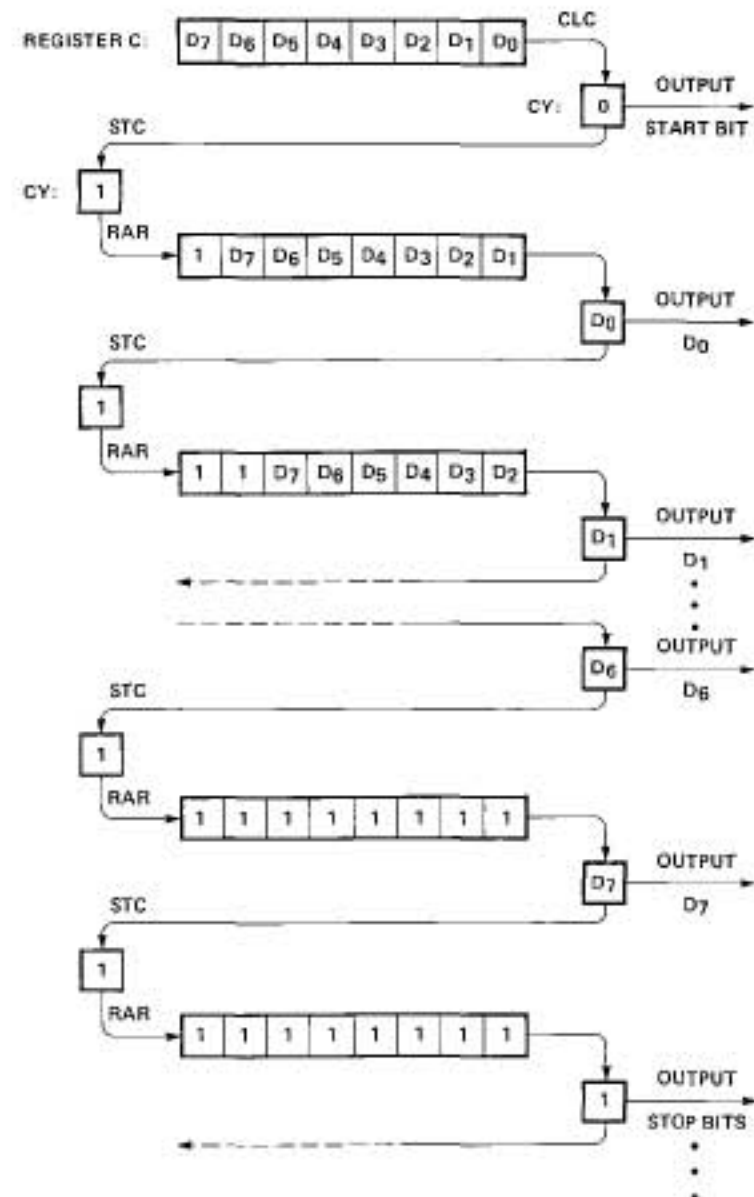


Figure 6. Data Serialization Algorithm

Get stuck in a loop for the appropriate time (don't worry for now how "BITTIME" is determined):

```
CO2:   LHL  BITTIME  <16>
        DCR  L        <4>
        JNZ  CO2     <4>
        DCR  H        <4>
        JNZ  CO2     <4>
```

Rotate the contents of register C right into the CY, while moving a one into the left end. Continue until all bits have been transmitted:

```
STC          <4>
MOV  A, C    <4>
RAR          <4>
MOV  C, A    <4>
DCR  B        <4>
JNZ  CO1     <10>
```

Restore processor status and return:

```

POP    H
POP    B
EI
RET

```

INPUT ROUTINE

The console input routine uses the opposite procedure; instead of moving a bit from register C to the CY, then to A₇, then to SOD, CIN loads a bit from SID into A₇, then moves it to CY, then into register C.

First, set up the CPU as before:

```

CIN:   PUSH    H
        DI
        MVI    B, 9

```

When a start bit transition arrives, the first sampling should not be taken until the middle of the first data bit, one and one-half bit times after the transition. Await the start bit transition, then set up the delay parameter for one-half bit time:

```

CI1:   RIM          <4>
        ORA    A          <4>
        JM     CI1        <7>
        LHLD   HALFBIT   <16>

```

Loop for one-half bit time before starting to sample data:

```

CI2:   DCR    L          <0>
        JNZ   CI2        <0>
        DCR    H          <0>
        JNZ   CI2        <0>

```

Wait until the middle of the next bit before sampling SID, then move the data bit into CY:

```

CI3:   LHLD   BITTIME   <16>
CI4:   DCR    L          <0>
        JNZ   CI4        <0>
        DCR    H          <0>
        JNZ   CI4        <0>
        RIM          <4>
        RAL          <4>

```

Decrement the bit counter. If this is the ninth cycle, the 8 data bits are in register C, so quit (the first stop bit will already have been received, and be in CY):

```

DCR    B          <4>
JZ     CI5        <7>

```

Otherwise, continue. Rotate the data bit right into register C, and repeat the cycle:

```

MOV    A, C      <4>
RAR          <4>
MOV    C, A      <4>
NOP          <4>
JMP    CI3       <10>

```

(A NOP is needed to make the COUT and CIN loops exactly equal in number of machine cycles, so that each can use the same delay parameter.) Restore status and return.

```

CI5:   POP    H
        EI
        RET

```

TIMING ANALYSIS

COUT and CIN now need to be provided with parameters for BITTIME and HALFBIT. It can be seen from the above code that each routine uses $61 + D$ machine cycles per input or output bit, where D is the number of cycles spent in either four line delay segment. If $\langle H \rangle$ and $\langle L \rangle$ are the contents of the H and L registers going into this section of code, then:

$$D = 22 + (\langle L \rangle - 1) \times 14 + (\langle H \rangle - 1) \times [(255 \times 14) + 25] \quad (1)$$

If $\langle H \rangle' \equiv \langle H \rangle - 1$, $\langle L \rangle' \equiv \langle L \rangle - 1$, and

$$\langle HL \rangle' \equiv 256 \langle H \rangle' + \langle L \rangle' \quad (2)$$

then

$$D = 22 + 14 \langle L \rangle' + 3595 \langle H \rangle' \quad (3)$$

This can be approximated by:

$$D = 22 + 14 \langle HL \rangle' \quad (4)$$

This approximation is exact for $\langle H \rangle' = 0$; otherwise, it is accurate to within 0.3%. Thus each loop of COUT or CIN uses a total of:

$$C = 61 + D = 83 + 14 \langle HL \rangle' \text{ machine cycles} \quad (5)$$

Each machine cycle uses two crystal cycles in the 8085, so the resulting data rate is:

$$B = \frac{\text{cycle frequency}}{C} = \frac{(\text{crystal frequency}) \div 2}{83 + 14 \langle HL \rangle'} \quad (6)$$

For a typical calculation, see Example 4.

EXAMPLE 4

To produce 2400 baud with the standard 6.144 MHz crystal:

$$2400 = \frac{(6.144 \times 10^6) \div 2}{83 + 14 \langle HL \rangle'}$$

$$14 \langle HL \rangle' = \left(\frac{6.144 \times 10^6 \div 2}{2400} \right) - 83$$

$$\langle HL \rangle' = \left[\left(\frac{6.144 \times 10^6 \div 2}{2400} \right) - 83 \right] \div 14 = 85.5 \cong 86$$

$$\langle HL \rangle' = 86_{10} = 0056H$$

$$\langle HL \rangle = 0157H = \text{BITTIME}$$

To determine the true data rate this parameter will produce, substitute into equation (6):

$$\text{Data Rate} = \frac{6.144 \times 10^6 \div 2}{83 + 14(86)}$$

$$= 2387 \text{ baud, which is 0.54\% slow.}$$

For 9600 baud, the same calculations will yield $\langle HL \rangle' = 17$, which is actually 0.3% slow; a sizzling 19200 baud or 38400 baud could each be generated to within 5% if $\langle HL \rangle' = 6$ or 0! Table 1 presents the parameters for several standard baud rates.

Notice that the resolution of the delay algorithm – the difference between bit times resulting from parameters which differ by one – is 14 machine cycles. As a result, the true bit delay produced can always manage to be within $\pm 2.3 \mu\text{sec}$ of the delay

desired. This guarantees that at rates up to 9600 baud, where each bit time is at least 104 μsec wide, some value of BITTIME can be found which will be accurate to within 2.2%.

BAUD RATE IDENTIFICATION ROUTINE

The function of BRID is to compute the appropriate parameters BITTIME and HALFBIT. It accomplishes this by observing the data pattern received when the space bar is pressed on the console device. Since a space character has the ASCII code 20H = 00100000B, the pattern represented back in Figure 4 is transmitted. Notice that the initial zero level is 6 bits wide. Suppose it could be determined that this corresponds to M machine cycles. Then one bit would correspond to (M÷6) machine cycles. The reason for dividing down a space several bits long is so that any distortion caused by the signal rise and fall times, or any lack of precision in detecting the two transitions, will be reduced by a factor of six. Since the bit period of COUT and CIN is 83 + 14 $\langle HL \rangle'$, BRID must generate a value $\langle HL \rangle'$ such that:

$$M \div 6 = 83 + 14 \langle HL \rangle' \tag{7}$$

$$\langle HL \rangle' = \frac{(M \div 6) - 83}{14} \tag{8}$$

$$\langle HL \rangle' = \frac{M}{84} - 6 \text{ (approximately)} \tag{9}$$

This value can be determined by setting register pair HL to -6, then incrementing it once every 84 machine cycles during the period that the incom-

Table 1

DELAY PARAMETERS FOR STANDARD BAND RATES USING 6.144 MHz CRYSTAL

TARGET BAUD RATE	$\langle HL \rangle'_{10}$ (See Text)	$\langle HL \rangle'_{16}$ (See Text)	$\langle HL \rangle$ or BITTIME (See Text)	HALFBIT	ACTUAL BAUD RATE PRODUCED	% ERROR
110	1989	07C5	08C6	04E3	109.99	-0.006
150	1457	05B1	06B2	03D9	149.99	-0.005
300	726	02D6	03D7	026C	299.80	-0.068
600	360	0168	0269	01A5	599.65	-0.059
1200	177	00B1	01B2	0159	1199.5	-0.039
2400	86	0056	0157	012C	2386.9	-0.547
4800	40	0028	0129	0115	4777.6	-0.469
9600	17	0011	0112	0109	9570.1	-0.312
19200	6	0006	0107	0104	18395.2	-4.37

ing signal is zero. BITTIME is then obtained by individually incrementing registers H and L. To obtain HALFBIT, divide the value of (HL) determined above by two before incrementing each register.

In order to implement this algorithm, set HL to -6, verify that the incoming signal is a logic one, then wait for the start bit transition.

```
BR10: MVI    A, 000H
      SIM
      LXI    H, -6H
BR11: RIM
      ORA    A
      JP     BR11
BR12: RIM
      ORA    A
      JM     BR12
```

Increment register pair HL, then delay so that each cycle will require 84 machine cycles:

```
BR13: INX    H           <6>
      MVI    E, 04H     <7>
BR14: DCR    E           <53>
      JNZ    BR14       <'>
```

Check if SID is still low. If so, repeat:

```
RIM           <4>
ORA    A      <4>
JP     BR13   <10>
```

Otherwise continue. Store HL temporarily for the HALFBIT calculation. Obtain and store BITTIME:

```
PUSH    H
INR     H
INR     L
SHLD    BITTIME
```

Restore HL, calculate HALFBIT, and return:

```
POP     H
ORA     A
MOV     A, H
RAR
MOV     H, A
MOV     A, L
RAR
MOV     L, A
INR     H
INR     L
SHLD    HALFBIT
RET
```

The assembled listings for these subroutines, along with a simple test program, is presented in the Appendix.

CASSETTE RECORDER INTERFACE

There are many situations where data has to be transmitted through a non-ideal medium. To give three typical examples, a system with electrically isolated elements might require that signals be AC coupled, communications through an audio network (such as telephone or radio) are greatly bandwidth limited, and some applications (such as a distributed network in an industrial environment) must tolerate random electrical noise. Attempting to record data on a cheap cassette recorder (the one used for this note cost \$17.00) will reveal all of these shortcomings, plus one: The tape speed fluctuates significantly and varies as the batteries run down, hence the data rate is inconsistent.

The recording scheme used here makes very few demands on the transmission medium. It makes no attempt to transmit DC voltage levels. Instead, data is transmitted by a series of variable length tone bursts. The dominant frequency of the tone used can be selected to be within the passband of the particular medium. Data is transmitted with each bit composed of a tone burst followed by a pause. The first third of a bit period is always a tone burst, the middle third is either a tone burst continuous with the first or a pause corresponding to, respectively, a one or zero, and the final third is always a pause, as shown in Figure 7. Thus, data is distinguished by the burst/pause ratio.

Hardware Design

These tone bursts are obtained from the 8085 SOD line, using analog signal conditioning to eliminate the DC component of the waveform. (This low frequency component is due to the single-ended nature of the SOD line: its deviations from ground are all positive, which unbalances the capacitive input stage of the recorder.) A suggested interface circuit is shown in Figure 8, using one LM324 quad op amp and a few standard value discrete components which should be available in even a digital design laboratory. On playback, analog circuitry is again used to detect the presence of a tone burst. In Figure 8, A2 buffers the incoming signal, and A3 inverts it. The peaks of these two signals are transmitted through D1 or D2 and are filtered by an RC network. Comparator A4 then squares up the output and produces the logic signal read

by the SID pin. Since the op amps are powered by the single 5-volt supply, a 2.0-volt reference level is obtained from a resistive voltage divider. The waveforms present at several points in the circuit are shown in Figure 9.

Software

The algorithm for reading a data bit off the tape is simple and straightforward: If the tone burst is longer than the pause, the bit is a one. Otherwise, it is a zero. Since only the time ratio is considered, any variation in tape speed will not affect the data determination.

VOLUME CONTROL

A question that arises with any audio cassette interface is how to set the volume control. (Recording level is usually determined internally.) When the playback level is correct, the logic signal output from A4 will have either a one-third or two-thirds duty cycle. This can be readily observed with an oscilloscope. In the field, an old-fashioned mechanical-type voltmeter could be connected to the A4 output, and the volume adjusted until the meter needle hovered somewhere between 1/3 and 2/3 the high level output voltage. With random data, the reading would be about 2 volts. There will be a fairly wide range of acceptable volume settings. (Since the quivering meter needle is being used here for inertial signal averaging, a digital voltmeter would not be very helpful in this application.)

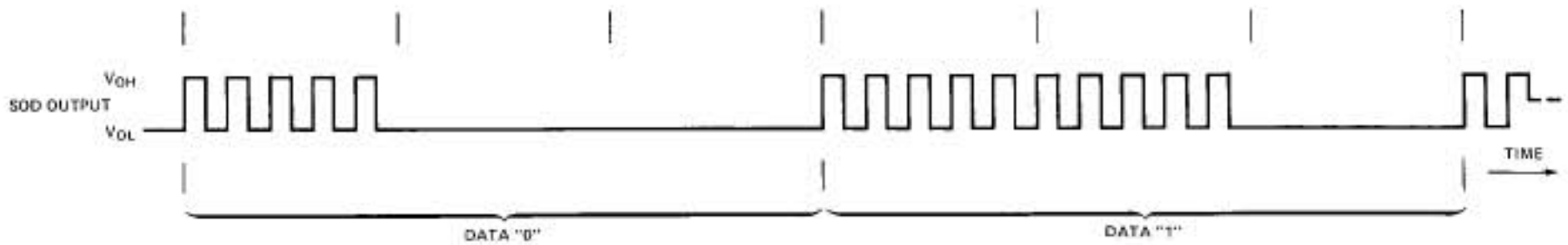


Figure 7. Tape Interface Data Recording Scheme

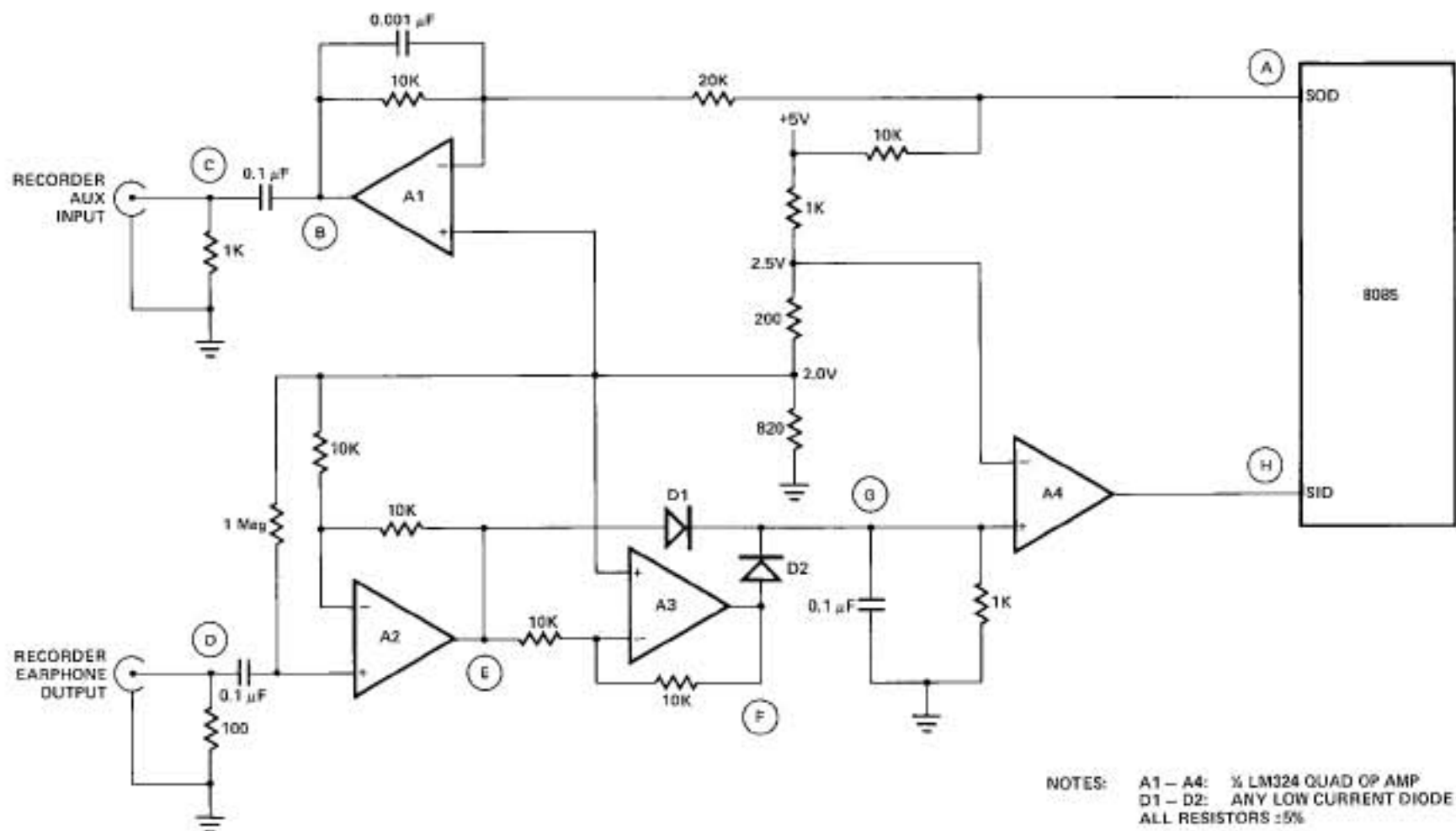


Figure 8. One Chip Magnetic Tape Interface Schematic

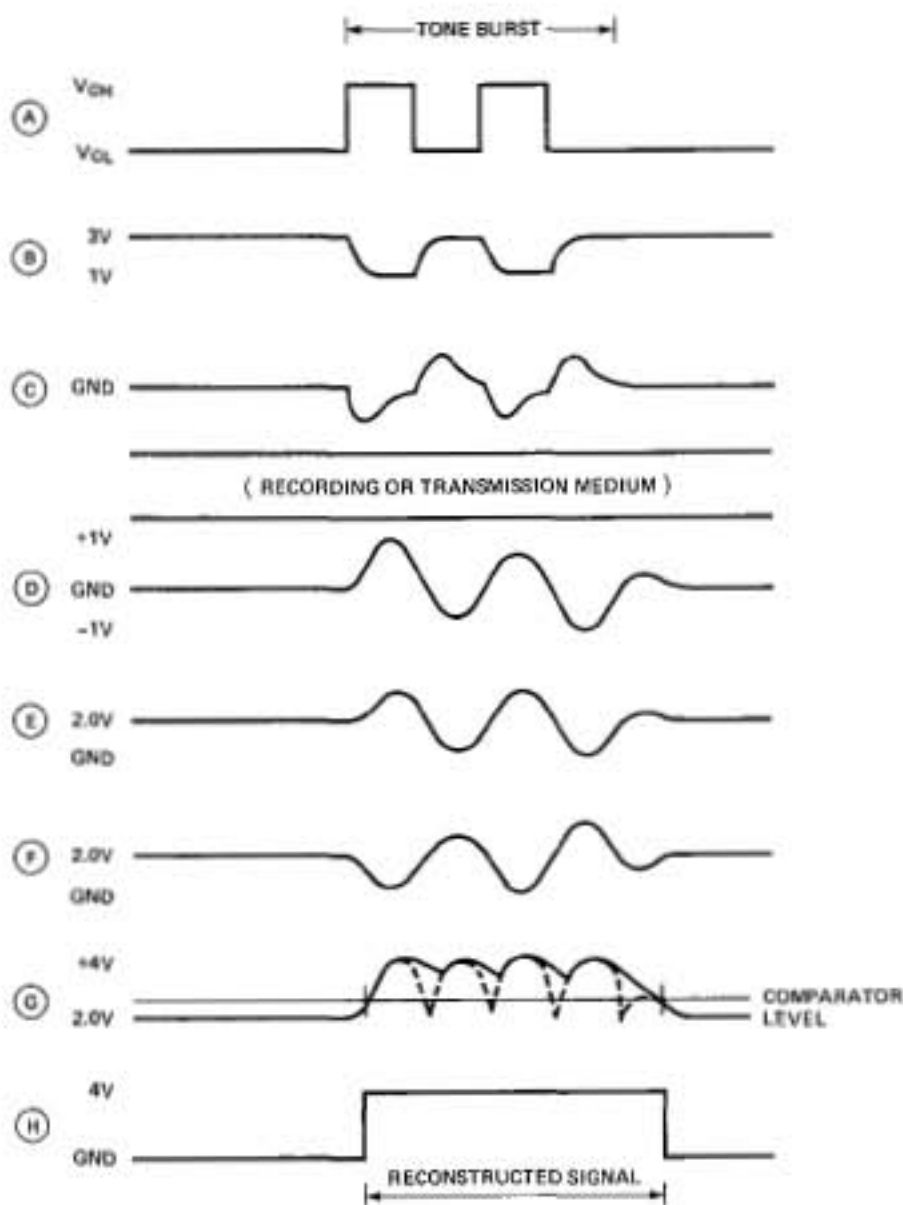


Figure 9. Analog Signal Waveforms

After the CRT software analysis, the tape routines are almost trivial. TAPEO is a subroutine for outputting the contents of register C to a cassette recorder. TAPEIN reads 8 bits into register C.

OUTPUT ROUTINE

TAPEO calls a subroutine named BURST three times for each bit. If A₆ (the SOD enable bit) is set when BURST is called, a square-wave tone burst will be transmitted. If A₆ is not set, BURST simply delays for exactly the same amount of time before returning. The three calls are used to, respectively, output the initial burst, output the data burst/space, and create the space at the end of each bit. Nine bits will be output: the eight data bits (LSB first) followed by a zero bit. The start of the initial burst of the trailing zero is needed to mark the end of the final space of the preceding data bit.

Start each bit by outputting a tone burst:

```
TAPEO: MVI    B, 9
T01:   MVI    A, 0C0H
      CALL   BURST
```

Rotate register C through CY:

```
MOV    A, C
RAR
MOV    C, A
```

Move CY to the SOD enable bit position, A₆. Simultaneously set A₇ to one, and clear all other bits. Output a tone burst or space, depending on the previous contents of CY:

```
MVI    A, 01H
RAR
RAR
CALL   BURST
```

Clear the accumulator, and output a space:

```
XRA    A
CALL   BURST
```

Keep cycling until the full 9-bit sequence is finished:

```
DCR    B
JNZ    T01
RET
```

The BURST subroutine executes the SIM instruction CYCNO times, at intervals of 29 + 14 (HALFCYC) machine cycles. In between each SIM, bit A₇ is complemented. CYCNO should be an even number. If A₆ is set upon calling BURST a square-wave will be created. Otherwise, the same code sequence is followed but SOD does not change – thus a space results.

```
BURST: MVI    D, CYCNO    <7>
BU1:   SIM
      MVI    E, HALFCYC  <7>
BU2:   DCR    E          <4>
      JNZ    BU2        <7/10>
      XRI    80H        <7>
      DCR    D          <4>
      JNZ    BU1        <7/10>
      RET                <10>
```

INPUT ROUTINE

TAPEIN uses a subroutine called BITIN to move the data at the SID pin into the CY. The maximum rate at which SID is read is limited by a delay loop in BITIN.

Initialize the bit counter and the register D, which will keep track of the tone burst time. If a tone

burst is being received when TAPEIN is called, wait until the burst is over:

```

TAPEIN: MVI    B, 8
        MVI    D, 00H
TI1:    CALL   BITIN
        JC     TI1
        CALL   BITIN
        JC     TI1
    
```

(Throughout this subroutine, a level transition is recognized only after it has been read once initially and then verified on the next reading. This provides some degree of software noise immunity.) Now await the start of the next burst:

```

TI2:    CALL   BITIN
        JNC   TI2
        CALL   BITIN
        JNC   TI2
    
```

The next burst has now arrived. Keep reading the SID pin, decrementing register D (thus making it more negative), each cycle until the pause is detected:

```

TI3:    DCR    D
        CALL   BITIN
        JC     TI3
        CALL   BITIN
        JC     TI3
    
```

Now continue reading the SID pin, incrementing the D register (back towards zero), each cycle until the next burst is received:

```

TI4:    INR    D
        CALL   BITIN
        JNC   TI4
        CALL   BITIN
        JNC   TI4
    
```

Now, if the burst lasted longer than the space, D was not incremented all the way back to zero; it is still negative. If the space was longer, D was incremented up through zero; it is now positive. In other words, the sign bit of D will now correspond to the data bit that would lead to each of these results. Move the sign bit into the CY, then rotate it into register C:

```

MOV    A, D
RAL
MOV    A, C
RAR
MOV    C, A
MVI    D, 00H
    
```

Continue until the last bit has been received:

```

DCR    B
JNZ    TI3
RET
    
```

(Notice that the first half of this subroutine is incorporated in the second half. In fact, the assembled listing included in the Appendix makes use of this fact to eliminate 24 bytes of duplicated code.)

BITIN waits a short time in order to regulate the sampling rate, then reads SID and moves the data bit into the CY:

```

BITIN: MVI    E, CKRATE    <7>
BI1:   DCR    E            <4>
        JNZ    BI1        <7/10>
        RIM            <4>
        RAL            <4>
        RET            <10>
    
```

The tone burst frequency and duration, and the TAPEIN sampling rate are determined by HALFCYC, CYCNO, and CKRATE. Tables 2 and 3 give typical values.

Table 2
EXAMPLE COMBINATIONS OF HALFCYC AND CYCNO.
ALL VALUES IN DECIMAL

APPROXIMATE TONE FREQUENCY	CORRESPONDING HALFCYC VALUE	RESULTING DATA RATE			
		8 4	20 10	100 50	CYCNO CYC/BURST
500 Hz	217	42	17	3.3	bps
1 kHz	108	83	33	6.6	bps
2 kHz	53	166	66	13	bps
5 kHz	20	414	166	33	bps
10 kHz	9	826	330	66	bps

Table 3
MAXIMUM SAMPLING RATES
FOR VARIOUS VALUES OF
CKRATE

CKRATE VALUE	SAMPLING RATE (INCLUDING CALL & RET)
1	17.6 μ sec
20	104 μ sec
80	378 μ sec
250	1.14 msec

The Appendix also includes a simple block record routine utilizing TAPEO. Before calling BLKRCD, HL must be set to the start of the desired block, and the recorder turned on manually. Successive bytes will be recorded until the end of that page, i.e., until L is incremented to zero. The playback routine requires presetting HL to the target address and turning on the recorder before PLAYBK is called. These routines incorporate a long tone burst before each data block to allow a recorder with Automatic Gain Control to stabilize before the data starts.

ADDITIONAL COMMENTS

The two design examples given so far were built up using an SDK-85 System Design Kit. Both hardware interfaces were wire-wrapped on the ample breadboarding area provided on the board. The connections between SID and SOD and the on-board TTY interface were broken, so as not to affect the 8085 I/O electrical characteristics.

The CRT interface was tested with a Beehive Mini-Bee II Terminal in the full duplex mode at each of its 14 possible transmission rates, from 110 to 9600 baud. It was also checked out at 19200 baud using a Beehive B-100 terminal. In addition, the software was exercised using an SBC 80/20 system as a variable baud rate character generator and receiver.

An additional advantage to having software selectable communications rates is that it would be possible to communicate with several system peripherals, each at its own preferred rate, without having to duplicate hardware. For example, the addition of a single 7408 AND gate and an output port (such as on the 8155) would make it possible to use the same two RS-232 circuits to interface with up to seven I/O devices (see Figure 10). Three of the MC1488 drivers have Enable inputs which can be controlled by the output port. One AND gate can be used to buffer the SOD line and drive the MC1488 Data inputs. The rest of the 7408 can be configured as a four input AND gate. This would act as an inverted logic OR gate to reduce the four MC1489 receiver outputs to a single line, which could be read by the SID. This assumes that only one input device (CRT, PTR) at a time will be used (which is usually the case in a non-time shared, interactive application), and that the unused devices are transmitting a logic one level (which should also be the case).

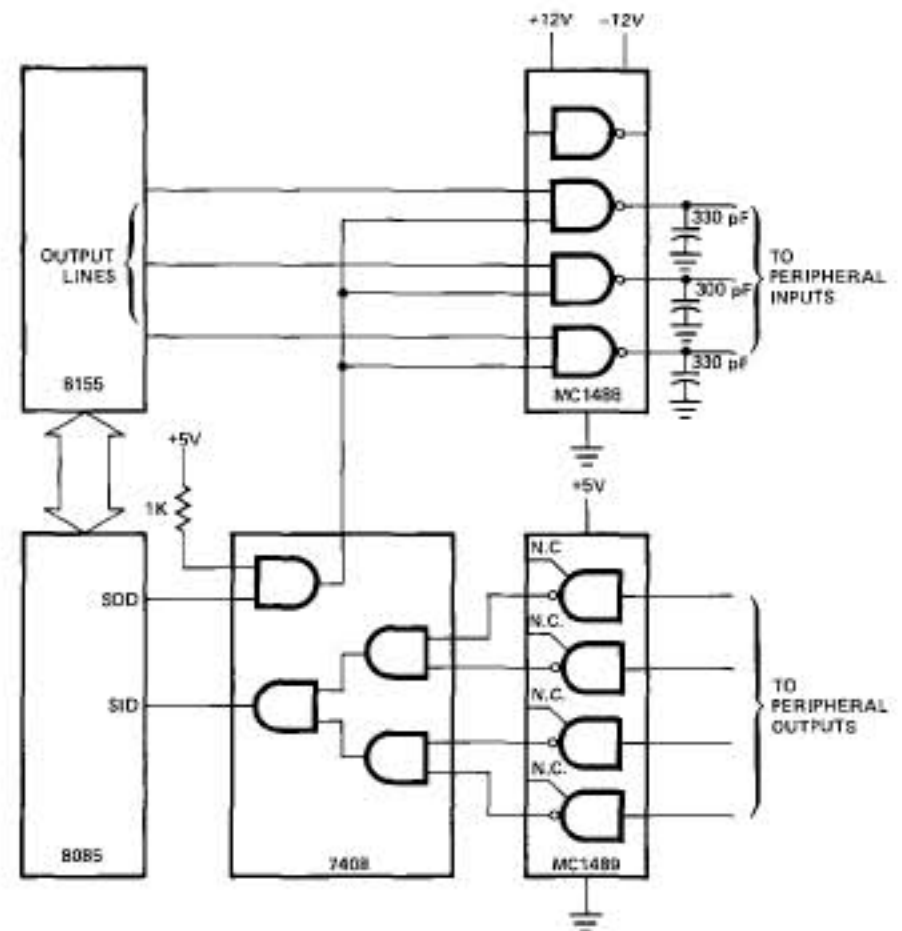


Figure 10. Interfacing 8085 to Multiple Peripherals

The software needed to support additional peripherals would be simple and straightforward. A routine intended to dump a section of memory to a paper tape punch, for example, would first have to store BITTIME and HALFBIT somewhere (perhaps on stack), load the variables with new parameters corresponding to the paper tape punch rate, and then write a bit pattern to the output port which would disable the console driver and enable the punch (and perhaps a typewriter). After the dump was over, the original time parameters and driver status would be restored.

As explained before, the BRID routine computed rate parameters based on the fact that an ASCII "space" character resulted in a zero level 6 bits long. Conceivably, some obscure peripherals might produce a transient between successive zero bits. (This might be the case, for example, if the signal was produced by mechanical rather than electronic means.) If so, the BRID algorithm used here probably would not work reliably. Once the two time parameters were identified, though, COUT and CIN could still be used. An alternate algorithm for baud rate identification would require a table in ROM (note the fifth and final R/S-I/O-M/D permutation). This table would contain a list of delay parameters corresponding to the standard transmis-

sion rates, as computed for the selected crystal frequency. Initialization would require the operator to hit a specific key several times (usually the "U" key, which generates a pattern of alternating ones and zeros). The identification routine would attempt to "read" this pattern at each baud rate, in turn, until finding the rate at which the read was successful.

The cassette recorder used to develop the tape interface was a Lloyd's push-button model which cost \$17 in 1972. Empirical testing has indicated that for this application, the quality of the cassette recorder is less critical than the quality of the tape itself. In other words, some 33¢ cassettes were not very reliable, even when used with more expensive recorders.

When using a cassette at the beginning of a side, allow the tape to run for about 10 seconds until the leader has passed before starting to write data. Otherwise, data will be lost to the leader.

Depending on the recorder quality, the tone burst frequency and duration can be optimized for higher data rates by modifying HALFCYC and CYCNO. If so, CKRATE should also be reduced, so that between about 10 and 80 data samplings are made during a single (one-third width) tone burst. At greatly increased frequencies, some of the

components in the analog interface might also be modified.

The two simple routines for recording and playing back blocks of data were intended to illustrate one way of using TAPEIN and TAPEO, and therefore do not contain any provisions for error detection or correction. Depending on the nature of a particular application, these routines could be augmented with parity bit or checksum comparison, or an error correcting code technique.

Funny things happen when recording and playing back a page of RAM which includes the subroutine stack. Eventually, PLAYBK will start writing over the data at the top of the stack, destroying the subroutine traceback sequence. The next RET instruction will then cause a jump to a place where you'd rather not be.

The printout reproduced in the Appendix includes the assembled listings for the CRT and magnetic tape interfaces discussed in this application note. The object code produced was programmed into an 8755 EPROM, which was installed in the expansion PROM socket of the SDK-85 board. Some very minor differences exist between this listing and the code segments presented earlier, which were written for maximum clarity.

APPENDIX

ISIS-II 8080/8085 ASSEMBLER, V1.0

MODULE

PAGE 1

LOC OBJ SEQ SOURCE STATEMENT

0 # MOD85 TITLE('8085 SERIAL I/O NOTE APPENDIX')

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	
		2 ;	THE FOLLOWING PROGRAMS AND SUBROUTINES ARE DESCRIBED IN DETAIL
		3 ;	IN INTEL CORPORATION'S APPLICATION NOTE AP-29, "USING THE 8085
		4 ;	SERIAL I/O LINES". THE FIRST SECTION IS A GENERAL PURPOSE CRT
		5 ;	INTERFACE WITH AUTOMATIC BAUD RATE IDENTIFICATION; THE SECOND
		6 ;	SECTION IS A MAGNETIC TAPE INTERFACE FOR STORING DATA ON CASSETTE
		7 ;	TAPES. THE CODE PRESENTED HERE IS ORIGINATED AT LOCATION 800H,
		8 ;	AND MIGHT BE PART OF AN EXPANSION PROM IN AN INTEL SDK-85
		9 ;	SYSTEM DESIGN KIT.
		10 ;	
		11	
		12	
20C8		13	BITTIME EQU 20C8H ; ADDRESS OF STORAGE FOR COMPUTED BIT DELAY
20CA		14	HALFBIT EQU 20CAH ; ADDRESS OF STORAGE FOR HALF BIT DELAY
000B		15	BITSO EQU 11 ; DATA BITS PUT OUT (INCLUDING TWO STOP BITS)
0009		16	BITSI EQU 9 ; DATA BITS TO BE RECEIVED (INCLUDING ONE STOP BIT)
		17	
0000		18	ORG 800H ; STARTING ADDRESS OF SDK-85 EXPANSION PROM
		19	
		20 ;	CRTTST CRT INTERFACE TEST. WHEN CALLED, AWAITS THE SPACE BAR BEING PRESSED ON
		21 ;	THE SYSTEM CONSOLE, AND THEN RESPONDS WITH A DATA RATE VERIFICATION
		22 ;	MESSAGE. THERE AFTER, CHARACTERS TYPED ON THE KEYBOARD ARE ECHOED
		23 ;	ON THE DISPLAY TUBE. WHEN A BREAK KEY IS TYPED, THE ROUTINE IS
		24 ;	RE-STARTED, ALLOWING A DIFFERENT BAUD RATE TO BE SELECTED ON THE CRT.
0000	31C020	25	CRTTST: LXI SP, 20C8H
0003	3EC0	26	CRT1: MVI A, 0C0H ; S0D MUST BE HIGH BETWEEN CHARACTERS
0005	30	27	SIM
0006	CD1A00	28	CALL BRID ; IDENTIFY DATA RATE USED BY TERMINAL
0009	CD4700	29	CALL SIGNON ; OUTPUT SIGNON MESSAGE AT RATE DETECTED
000C	CD8A00	30	ECHO: CALL CIN ; READ NEXT KEYSTROKE INTO REGISTER C
000F	79	31	MOV A, C
0010	B7	32	ORA A ; CHECK IF CHARACTER WAS A <BREAK> (ASCII 00H)
0011	CA0300	33	JZ CRT1 ; IF SO, RE-IDENTIFY DATA RATE
		34	; THIS ALLOWS ANOTHER RATE TO BE SELECTED ON CRT
0014	CD6900	35	CALL COUT ; OTHERWISE COPY REGISTER C TO THE SCREEN
0017	C30C00	36	JMP ECHO ; CONTINUE INDEFINITELY (UNTIL BREAK)
		37	
		38 ;	BRID BAUD RATE IDENTIFICATION SUBROUTINE
		39 ;	EXPECTS A <CR> (ASCII 20H) TO BE RECEIVED FROM THE CONSOLE.
		40 ;	THE LENGTH OF THE INITIAL ZERO LEVEL (SIX BITS WIDE) IS MEASURED
		41 ;	IN ORDER TO DETERMINE THE DATA RATE FOR FUTURE COMMUNICATIONS.
001A	20	42	BRID: RIM ; VERIFY THAT THE "ONE" LEVEL HAS BEEN ESTABLISHED
001B	B7	43	ORA A ; \ AS THE CRT IS POWERING UP
001C	F21A00	44	JP BRID
001F	20	45	BRI1: RIM ; MONITOR SID LINE STATUS
0020	B7	46	ORA A
0021	FA1F00	47	JM BRI1 ; LOOP UNTIL START BIT IS RECEIVED
0024	21FAFF	48	LXI H, -6 ; BIAS COUNTER USED IN DETERMINING ZERO DURATION
0027	1E04	49	BRI3: MVI E, 04H
0029	10	50	BRI4: DCR E ; 53 MACHINE CYCLE DELAY LOOP
002A	C22900	51	JNZ BRI4
002D	23	52	INX H ; INCREMENT COUNTER EVERY 84 CYCLES WHILE SID IS LOW
002E	20	53	RIM

LOC	OBJ	SEQ	SOURCE STATEMENT
082F	B7	54	ORA A
0830	F22708	55	JP BR13
		56	; <HL> NOW CORRESPONDS TO INCOMING DATA RATE
0833	E5	57	PUSH H ; SAVE COUNT FOR HALFBIT TIME COMPUTATION
0834	24	58	INR H ; BITTIME IS DETERMINED BY INCREMENTING
0835	2C	59	INR L ; \ H AND L INDIVIDUALLY
0836	22C820	60	SHLD BITTIME
0839	E1	61	POP H ; RESTORE COUNT FOR HALFBIT DETERMINATION
083A	B7	62	ORA A ; CLEAR CARRY
083B	7C	63	MOV A,H ; ROTATE RIGHT EXTENDED <HL>
083C	1F	64	RAR ; \ TO DIVIDE COUNT BY 2
083D	67	65	MOV H,A
083E	7D	66	MOV A,L
083F	1F	67	RAR
0840	6F	68	MOV L,A
0841	24	69	INR H ; PUT H AND L IN PROPER FORMAT FOR DELAY
0842	2C	70	INR L ; \ SEGMENTS (INCREMENT EACH)
0843	22CA20	71	SHLD HALFBIT ; SAVE AS HALF-BIT TIME DELAY PARAMETER
0846	C9	72	RET
		73	
		74	; SIGNON WRITES A SIGN-ON MESSAGE TO THE CRT AT WHAT SHOULD BE THE CORRECT RATE.
		75	; IF THE MESSAGE IS UNINTELLIGIBLE . . . WELL, SO IT GOES.
0847	215508	76	SIGNON: LXI H,STRNG ; LOAD START OF SIGN-ON MESSAGE
084A	4E	77	S1: MOV C,M ; GET NEXT CHARACTER
084B	AF	78	XRA A ; CLEAR ACCUMULATOR
084C	B1	79	ORA C ; CHECK IF CHARACTER IS END OF STRING
084D	C8	80	RZ ; RETURN IF SIGN-ON COMPLETE
084E	CD6908	81	CALL COUT ; ELSE OUTPUT CHARACTER TO CRT
0851	23	82	INX H ; INDEX POINTER
0852	C34A08	83	JMP S1 ; ECHO NEXT CHARACTER
		84	
0855	0D	85	STRNG: DB 0DH,0AH ; <CR><LF>
0856	0A		
0857	42415544	86	DB 'BAUD RATE CHECK'
085B	20524154		
085F	45204348		
0863	454348		
0866	0D	87	DB 0DH,0AH ; <CR><LF>
0867	0A		
0868	0D	88	DB 0DH ; END-OF-STRING ESCAPE CODE
		89	
		90	; COUT CONSOLE OUTPUT SUBROUTINE
		91	; WRITES THE CONTENTS OF THE C REGISTER TO THE CRT DISPLAY SCREEN
0869	F3	92	COUT: DI
086A	C5	93	PUSH B
086B	E5	94	PUSH H
086C	0608	95	MVI B,BIT50 ; SET NUMBER OF BITS TO BE TRANSMITTED
086E	AF	96	XRA A ; CLEAR CARRY
086F	3E80	97	C01: MVI A,80H ; SET WHAT WILL BECOME S00 ENABLE BIT
0871	1F	98	RAR ; MOVE CARRY INTO S00 DATA BIT OF ACC
0872	30	99	SIM ; OUTPUT DATA BIT TO S00
0873	2AC820	100	LHLD BITTIME
0876	2D	101	C02: DCR L ; WAIT UNTIL APPROPRIATE TIME HAS PASSED

LOC	OBJ	SEQ	SOURCE STATEMENT
0877	C27608	102	JNZ C02
087A	25	103	DCR H
087B	C27608	104	JNZ C02
087E	37	105	STC ;SET WHAT WILL EVENTUALLY BECOME A STOP BIT
087F	79	106	MOV A,C ;ROTATE CHARACTER RIGHT ONE BIT,
0880	1F	107	RAR ;\ MOVING NEXT DATA BIT INTO CARRY
0881	4F	108	MOV C,A
0882	05	109	DCR B ;CHECK IF CHARACTER (AND STOP BIT(S)) DONE
0883	C26F08	110	JNZ C01 ;IF NOT, OUTPUT CURRENT CARRY
0886	E1	111	POP H ;RESTORE STATUS AND RETURN
0887	C1	112	POP B
0888	FB	113	EI
0889	C9	114	RET
		115	
		116	;CIN CONSOL INPUT SUBROUTINE WAITS FOR A KEYSTROKE AND
		117	; RETURNS WITH 8 BITS IN REG C.
088A	F3	118	CIN: DI
088B	E5	119	PUSH H
088C	0609	120	MVI B,BITSI ;DATA BITS TO BE READ (LAST RETURNED IN CY)
088E	20	121	CI1: RIM ;WAIT FOR SYNC BIT TRANSITION
088F	B7	122	ORA A
0890	FA8E08	123	JM CI1
0893	2ACA20	124	LHLD HALFBIT
0896	20	125	CI2: DCR L ;WAIT UNTIL MIDDLE OF START BIT
0897	C29608	126	JNZ CI2
089A	25	127	DCR H
089B	C29608	128	JNZ CI2
089E	2AC820	129	CI3: LHLD BITTIME ;WAIT OUT BIT TIME
08A1	20	130	CI4: DCR L
08A2	C2A108	131	JNZ CI4
08A5	25	132	DCR H
08A6	C2A108	133	JNZ CI4
08A9	20	134	RIM ;CHECK SID LINE LEVEL
08AA	17	135	RAL ;DATA BIT IN CY
08AB	05	136	DCR B ;DETERMINE IF THIS IS FIRST STOP BIT
08AC	0A8608	137	JZ CI5 ;IF SO, JUMP OUT OF LOOP
08AF	79	138	MOV A,C ;ELSE ROTATE INTO PARTIAL CHARACTER IN C
08B0	1F	139	RAR ;ACC HOLDS UPDATED CHARACTER
08B1	4F	140	MOV C,A
08B2	00	141	NOP ;EQUALIZES COUT AND CIN LOOP TIMES
08B3	C29E08	142	JMP CI3
08B6	E1	143	CI5: POP H
08B7	FB	144	EI
08B8	C9	145	RET ;CHARACTER COMPLETE
		146	
		147	;*****
		148	
		149	; THE FOLLOWING CODE IS USED BY THE CASSETTE INTERFACE.
		150	; SUBROUTINES TAPEO AND TAPEIN ARE USED RESPECTIVELY
		151	; TO OUTPUT OR RECEIVE AN EIGHT BIT BYTE OF DATA REGISTER C
		152	; HOLDS THE DATA IN EITHER CASE. REGISTERS A,B,&C ARE ALL DESTROYED.
0010		153	CYCNO EQU 16 ;TWICE THE NUMBER OF CYCLES PER TONE BURST
001E		154	HALFCYC EQU 30 ;DETERMINES TONE FREQUENCY

LOC	OBJ	SEQ	SOURCE	STATEMENT
0016		155	CKRATE EQU	22 ; SETS SAMPLE RATE
00FA		156	LEADER EQU	250 ; NUMBER OF SUCCESSIVE TONE BURSTS COMPRISING LEADER
00FA		157	LDRCHK EQU	250 ; USED IN PLAYBK TO VERIFY PRESENCE OF LEADER
		158		
		159	;BLKRCO	OUTPUTS A VERY LONG TONE BURST (<LEADER> TIMES
		160		THE NORMAL BURST DURATION) TO ALLOW RECORDER ELECTRONICS
		161		AND AGC TO STABILIZE, THEN OUTPUTS THE REMAINDER OF THE
		162		256 BYTE PAGE POINTED TO BY <H>, STARTING AT BYTE <L>.
00B9	0EFA	163	BLKRCO: MVI	C, LEADER; SET UP LEADER BURST LENGTH
00BB	3EC0	164		MVI A, 0C0H ; SET ACCUMULATOR TO RESULT IN TONE BURST
00BD	0DF008	165	BR1:	CALL BURST ; OUTPUT TONE
00C0	0D	166		DCR C
00C1	C2BD08	167		JNZ BR1 ; SUSTAIN LEADER TONE
00C4	AF	168		XRA A ; CLEAR ACCUMULATOR & OUTPUT SPACE, SO THAT
00C5	0DF008	169		CALL BURST ; \ START OF FIRST DATA BYTE CAN BE DETECTED
00C8	4E	170	BR2:	MOV C, M ; GET DATA BYTE TO BE RECORDED
00C9	CDD108	171		CALL TAPEO ; OUTPUT REGISTER C TO RECORDER
00CC	2C	172		INR L ; POINT TO NEXT BYTE
00CD	C2C008	173		JNZ BR2
00D0	C9	174		RET ; AFTER BLOCK IS COMPLETE
		175		
		176		
		177	;TAPEO	OUTPUTS THE BYTE IN REGISTER C TO THE RECORDER.
		178		REGISTERS A, B, C, D, & E ARE ALL USED.
00D1	F3	179	TAPEO: DI	
00D2	D5	180		PUSH D ; D&E USED AS COUNTERS BY SUBROUTINE BURST
00D3	0609	181		MVI B, 9 ; WILL RESULT IN 8 DATA BITS AND ONE STOP BIT
00D5	AF	182	T01:	XRA A ; CLEAR ACCUMULATOR
00D6	3EC0	183		MVI A, 0C0H ; SET ACCUMULATOR TO CAUSE A TONE BURST
00D8	0DF008	184		CALL BURST
00DB	79	185		MOV A, C ; MOVE NEXT DATA BIT INTO THE CARRY
00DC	1F	186		RAR
00DD	4F	187		MOV C, A ; CARRY WILL BECOME S0D ENABLE IN BURST ROUTINE
00DE	3E01	188		MVI A, 01H ; SET BIT TO BE REPEATEDLY COMPLEMENTED IN BURST
00E0	1F	189		RAR
00E1	1F	190		RAR
00E2	0DF008	191		CALL BURST ; OUTPUT EITHER A TONE OR A PAUSE
00E5	AF	192		XRA A ; CLEAR ACCUMULATOR
00E6	0DF008	193		CALL BURST ; OUTPUT PAUSE
00E9	05	194		DCR B
00EA	C2D508	195		JNZ T01 ; REPEAT UNTIL BYTE FINISHED
00ED	D1	196		POP D ; RESTORE STATUS AND RETURN
00EE	FB	197		EI
00EF	C9	198		RET
		199		
00F0	1610	200	BURST:	MVI D, CYCNO ; SET NUMBER OF CYCLES
00F2	30	201	BU1:	SIM ; COMPLEMENT S0D LINE IF S0D ENABLE BIT SET
00F3	1E1E	202		MVI E, HALFCYC
00F5	1D	203	BU2:	DCR E ; REGULATE TONE FREQUENCY
00F6	C2F508	204		JNZ BU2
00F9	EE00	205		XRI 00H ; COMPLEMENT S0D DATA BIT IN ACCUMULATOR
00FB	15	206		DCR D
00FC	C2F208	207		JNZ BU1 ; CONTINUE UNTIL BURST (OR EQUIVILENT PAUSE) FINISHED

LOC	OBJ	SEQ	SOURCE STATEMENT
08FF	C9	208	RET
		209	
		210	; PLAYBK
		211	; WAITS FOR THE LONG LEADER BURST TO ARRIVE, THEN CONTINUES
		212	; READING BYTES FROM THE RECORDER AND STORING THEM
		213	; IN MEMORY STARTING AT LOCATION <HL>.
		213	; CONTINUES UNTIL THE END OF THE CURRENT PAGE (<L>=0FFH) IS REACHED.
0900	0EFA	214	PLAYBK: MVI C, LDRCHK ; <LDRCHK> SUCCESSIVE HIGHS MUST BE READ
0902	CD3009	215	PB1: CALL BITIN ; \ TO VERIFY THAT THE LEADER IS PRESENT
0905	D20009	216	JNC PLAYBK ; \ AND ELECTRONICS HAS STABILIZED
0908	0D	217	DCR C
0909	C20209	218	JNZ PB1
090C	CD1509	219	PB2: CALL TAPEIN ; GET DATA BYTE FROM RECORDER
090F	71	220	MOV M, C ; STORE IN MEMORY
0910	2C	221	INR L ; INCREMENT POINTER
0911	C20C09	222	JNZ PB2 ; REPEAT FOR REST OF CURRENT PAGE
0914	C9	223	RET
		224	
		225	; TAPEIN CASSETTE TAPE INPUT SUBROUTINE. READS ONE BYTE OF DATA
		226	; FROM THE RECORDER INTERFACE AND RETURNS WITH THE BYTE IN REGISTER C.
0915	0609	227	TAPEIN: MVI B, 9 ; READ EIGHT DATA BITS
0917	1609	228	TI1: MVI D, 00H ; CLEAR UP/DOWN COUNTER
0919	15	229	TI2: DCR D ; DECREMENT COUNTER EACH TIME ONE LEVEL IS READ
091A	CD3009	230	CALL BITIN
091D	DA1909	231	JC TI2 ; REPEAT IF STILL AT ONE LEVEL
0920	CD3009	232	CALL BITIN
0923	DA1909	233	JC TI2
0926	14	234	TI3: INR D ; INCREMENT COUNTER EACH TIME ZERO IS READ
0927	CD3009	235	CALL BITIN
092A	D22609	236	JNC TI3 ; REPEAT EACH TIME ZERO IS READ
092D	CD3009	237	CALL BITIN
0930	D22609	238	JNC TI3
0933	7A	239	MOV A, D
0934	17	240	RAL ; MOVE COUNTER MOST SIGNIFICANT BIT INTO CARRY
0935	79	241	MOV A, C
0936	1F	242	RAR ; MOVE DATA BIT RECEIVED (CY) INTO BYTE REGISTER
0937	4F	243	MOV C, A
0938	05	244	DCR B
0939	C21709	245	JNZ TI1 ; REPEAT UNTIL FULL BYTE ASSEMBLED
093C	C9	246	RET
		247	
093D	1E16	248	BITIN: MVI E, CKRATE
093F	1D	249	BI1: DCR E
0940	C23F09	250	JNZ BI1 ; LIMIT INPUT SAMPLING RATE
0943	20	251	RIM ; SAMPLE SID LINE
0944	17	252	RAL ; MOVE DATA INTO CY BIT
0945	C9	253	RET
		254	
		255	END

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS

BI1	A 093F	BITIN	A 0930	BITSI	A 0009	BITSO	A 000B	BITTIM	A 20C8	BLKROD	A 08B9	BR1	A 0880
BR2	A 08C8	BRI1	A 081F	BRI3	A 0827	BRI4	A 0829	BRID	A 081A	BU1	A 08F2	BU2	A 08F5
BURST	A 08F0	CI1	A 088E	CI2	A 0896	CI3	A 089E	CI4	A 08A1	CI5	A 08B6	CIN	A 088A
CKRATE	A 0016	CO1	A 086F	CO2	A 0876	COU	A 0869	CRT1	A 0803	CRTTST	A 0800	CYCNO	A 0010
ECHO	A 080C	HALFBI	A 20CA	HALFCY	A 001E	LDRCHK	A 00FA	LEADER	A 00FA	PB1	A 0902	PB2	A 090C
PLAYBK	A 0900	S1	A 084A	SIGNON	A 0847	STRNG	A 0855	TAPEIN	A 0915	TAPEO	A 08D1	TI1	A 0917
TI2	A 0919	TI3	A 0926	T01	A 0805								

ASSEMBLY COMPLETE, NO ERROR(S)

BI1	249#	250				
BITIN	215	230	232	235	237	248#
BITS1	16#	120				
BITS0	15#	95				
BITTIM	13#	60	100	129		
BLKRCO	163#					
BR1	165#	167				
BR2	170#	173				
BRI1	45#	47				
BRI3	49#	55				
BRI4	50#	51				
BRID	28	42#	44			
BU1	201#	207				
BU2	203#	204				
BURST	165	169	184	191	193	200#
CI1	121#	123				
CI2	125#	126	128			
CI3	129#	142				
CI4	130#	131	133			
CI5	137	143#				
CIN	30	118#				
CKRATE	155#	248				
CO1	97#	110				
CO2	101#	102	104			
COUT	35	81	92#			
CRT1	26#	33				
CRTTST	25#					
CYCNO	153#	200				
ECHO	30#	36				
HALFBI	14#	71	124			
HALFCY	154#	202				
LDRCHK	157#	214				
LEADER	156#	163				
PB1	215#	218				
PB2	219#	222				
PLAYBK	214#	216				
S1	77#	83				
SIGNON	29	76#				
STRNG	76	85#				
TAPEIN	219	227#				
TAPEO	171	179#				
TI1	228#	245				
TI2	229#	231	233			
TI3	234#	236	238			
TO1	182#	195				

CROSS REFERENCE COMPLETE

Related Intel Publications

"8085 Microcomputer Systems User's Manual"

"8080/8085 Assembly Language Programming Manual"

The material in this Application Note is for informational purposes only and is subject to change without notice. Intel Corporation has made an effort to verify that the material in this document is correct. However, Intel Corporation does not assume any responsibility for errors that may appear in this document.

The following are trademarks of Intel Corporation and may be used only to describe Intel Products:

MCS
SDK-85
ICE-85



INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 (408) 987-8080

Printed in U.S.A./T97/0578/20K BL