PRENTICE-HALL SERIES IN PERSONAL COMPUTING

JOHN E. UFFENBECK

# Hardware Interfacing with the TRS-80

*A step by step introduction to microcomputer interfacing—featuring 14 practical hardware experiments.*

# HARDWARE INTERFACING WITH THE TRS-80

# HARDWARE INTERFACING WITH THE TRS-80

JOHN E. UFFENBECK

PRENTICE-HALL, INC., *Englewood Cliffs, NJ 07632*

**Library of Congress Cataloging in Publication Data**

Uffenbeck, John E. (date)
Hardware interfacing with the TRS-80.

Includes index.
1. Computer interfaces. 2. TRS-80 (Computer)
I. Title.
TK7887.5.U33 1983 621.3819'5832 82-13251
ISBN 0-13-383877-3
ISBN 0-13-383869-2 (pbk.)

*Editorial/production supervision*
*and interior design:* Karen Skrable
*Manufacturing buyer:* Gordon Osbourne
*Cover design:* Diane Saxe
Cover photo courtesy of Radio Shack.

© 1983 by Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632

*All rights reserved. No part of this book may be reproduced in any form or by any means without permission in writing from the publisher.*

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-383877-3 {CASE}
ISBN 0-13-383869-2 {PBK.}

Prentice-Hall International, Inc., *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Prentice-Hall of Southeast Asia Pte. Ltd., *Singapore*
Whitehall Books Limited, *Wellington, New Zealand*

# CONTENTS

PREFACE
*vii*

PART 1
GETTING STARTED
*1*

Introduction *3*
Essential Components *5*
Nice-to-Have Components *9*
Conclusion *11*

PART 2
BASIC CONCEPTS OF MICROCOMPUTER INPUT/OUTPUT
*13*

Experiment 1 The Four I/O Commands in BASIC *15*
Experiment 2 Address Decoding *25*
Experiment 3 I/O-Mapped Output Port Concepts *35*
Experiment 4 I/O-Mapped Input Port Concepts *46*
Experiment 5 I/O-Mapped I/O Using the 8255 *56*
Experiment 6 Memory-Mapped I/O Using the 8255 *69*
Experiment 7 1K Static Memory Interface *81*

## PART 3
## SPECIAL INTERFACING PROBLEMS

*91*

Experiment 8 Hardware Interfacing Techniques, Part 1: Inputs *93*
Experiment 9 Hardware Interfacing Techniques, Part 2: Outputs *106*
Experiment 10 Interfacing a Digital-to-Analog Converter *119*
Experiment 11 Interfacing an Analog-to-Digital Converter *131*
Experiment 12 Handshaking I/O *147*
Experiment 13 Serial Interfacing *163*
Experiment 14 Interfacing a Programmable Sound Generator *185*

## APPENDICES

*205*

A Description of the Model I and Model III Expansion Connectors *207*
B Parts List *213*
C Binary and Decimal Numbers *217*
D Basic Logic Gates *220*
E JK Flip-Flop *223*
F If the Experiment Doesn't Work *225*

## INDEX

*229*

# PREFACE

This is a *doing* book. It features *14* experiments on interfacing the outside world to the Radio Shack Model I and Model III TRS-80 computers. You will learn interfacing by *doing* interfacing. Each experiment is intended to be built on a breadboard and not constructed as a permanent circuit. The emphasis is on understanding input/output (I/O) concepts, not on wiring skills.

All the interface circuits will be controlled from BASIC and numerous annotated programs are included. As such, you should have had two- to three months of experience working with TRS-80 BASIC and writing your own programs. You will also need an understanding of the binary number system and be able to convert back and forth easily between binary and decimal. Most of the circuits will use 7400 family ICs, and a basic knowledge of flip-flops and logic gates is presumed. Appendices C, D, and E provide a review of these areas.

This book is organized into three parts. *Part 1* tells you what you need to get started. This is critical, as you must have certain hardware available to do the experiments. Fortunately, most of these components should be available to you locally at minimal cost. Appendix B gives a complete parts list and a key to several suppliers.

*Part 2* details the basic concepts of I/O that you must understand. The three-bus system architecture is introduced and simple input and output ports tested. Finally, the *8255* programmable peripheral interface chip is described and used for most of the remaining experiments.

Much of the outside world presents a "hostile" environment to the TRS-80 and *Part 3* describes interfacing to this non-TTL world. Included are special interfacing problems such as controlling 110-V AC devices, analog inputs and outputs, serial interfacing, and devices requiring "handshaking" techniques. The last experiment includes an interface to the General Instruments *AY-3-8910* programmable sound generator. This circuit allows a fantastic range of sounds to be produced by your TRS-80.

Each experiment is organized into five parts. In the *OVERVIEW* a quick description of what you will be doing in that experiment is given. Next, the experiment learning *OBJECTIVES* are listed. If you already feel comfortable about these objectives, you can then go on to another experiment. The *PARTS LIST* indicates all components needed to perform that experiment. A brief *DISCUSSION* follows, explaining the theory about to be tested on the breadboard. Examples help to illustrate the concepts being discussed. The *PROCEDURE* section details a step-by-step procedure for performing the experiment. Appendix E is provided to give troubleshooting hints "*if the experiment doesn't work.*"

So, if you are like many users and owners of microcomputers today and have never connected a piece of hardware to your computer system, be prepared for a pleasant surprise. You may find hardware interfacing every bit as fascinating and challenging as that last BASIC program you wrote.

I would like to thank Mark for helping to test these experiments, Karen for the photography, and my wife Kathy for understanding why I spent every night at the office for the last year. I dedicate this book to Benji and Michael.

JOHN E. UFFENBECK

# 1 Getting Started

Read this section before attempting any of the experiments in this book. You will need special hardware to do the experiments and this section tells you what to get and where you can find it.

# INTRODUCTION

To take full advantage of the various experiments in this book, there are certain hardware and software requirements for your computer system.

## MODEL I VERSUS MODEL III

Radio Shack no longer produces the Model I TRS-80, having replaced it with the newer Model III (see Fig. I-1). The main difference between these two computers, as far as this book is concerned, is that *memory-mapped I/O* cannot be done on the Model III. This is because the *high-order* address lines have not been brought out to the expansion connector on the underside of the Model III computer. Two of the experiments in this book require these high-order address lines and therefore *cannot* be done with the Model III. However, you may still wish to read through the discussion of these experiments to familiarize yourself with the concepts.

With the exception of these two experiments, all experiments in this book will work on *both* the Model I and Model III computers. Any special considerations that must be made to make an experiment work on one model or the other will be noted.

Regardless of your computer model, you should have a version of BASIC that includes the **INP**, **OUT**, **PEEK**, and **POKE** commands. This version of BASIC is called *Level II* on the Model I and *Model III* BASIC on the Model III. As long as your version of BASIC has these four commands, you should have no problems with software.

## HARDWARE

This book deals with computer *hardware* (integrated circuits, light-emitting diodes, switches, wire, etc.). Your TRS-80 computer will talk to this hardware via a multiconductor *umbilical cord* or ribbon cable. This cable is discussed in detail in the next section, "Essential Components."

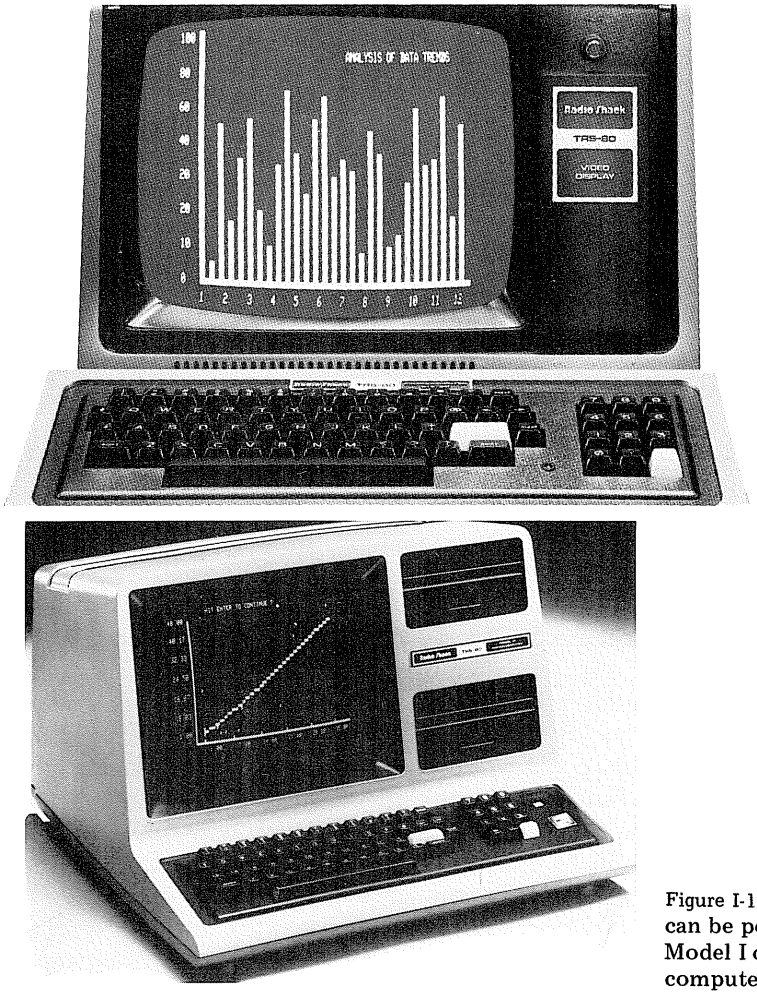



Figure I-1 Experiments in this book can be performed using (a) the Model I or (b) the Model III TRS-80 computer. (Courtesy of Radio Shack.)

In addition to this ribbon cable, you will need a *solderless breadboard* to construct the various hardware interfaces, a +5-V DC *power supply* to power these circuits (the TRS-80 does not have enough extra power to accomplish this), a supply of components, and various lengths of No. 22 or 24 gauge wire to perform each experiment.

Although you may be dismayed to learn that you must purchase these items, consider them to be the "tools of your trade" if you plan to be seriously involved in hardware interfacing. The next section will give you some hints on where to find these items and what possible substitutions can be made.

All the items mentioned so far must be considered essential hardware.

If you do not have these items, you will be *unable* to perform the majority of the experiments, if any. Of course, you may use this book as a guide to hardware interfacing and simply read through those experiments of interest to you, but maximum benefit is obtained by actually "getting your hands on the hardware."

There are also a couple of *nice-to-have* hardware items, such as a logic probe and a simple DC voltmeter. More about these in a later section.

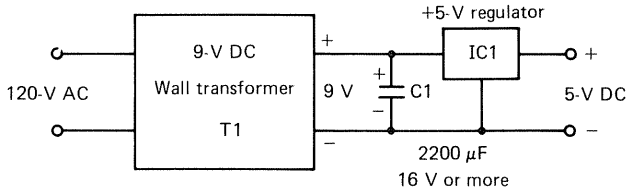
## ESSENTIAL COMPONENTS

In this section we discuss components that you must have available if you are to seriously attempt the experiments in the following two parts of this book.

### 5-V DC POWER SUPPLY

Nearly all the integrated circuits used in this book require +5 V DC for their function. When choosing a power supply, be sure it has at least a +5-V output rated at *0.2 A* of current or more. Although this will be sufficient for a majority of the experiments, it would also be desirable to have a *negative* output voltage (- 5 V) for a couple of the experiments.

Fully assembled power supplies are readily available, but it may be cheaper to assemble your own. Figure I-2 illustrates a simple power supply

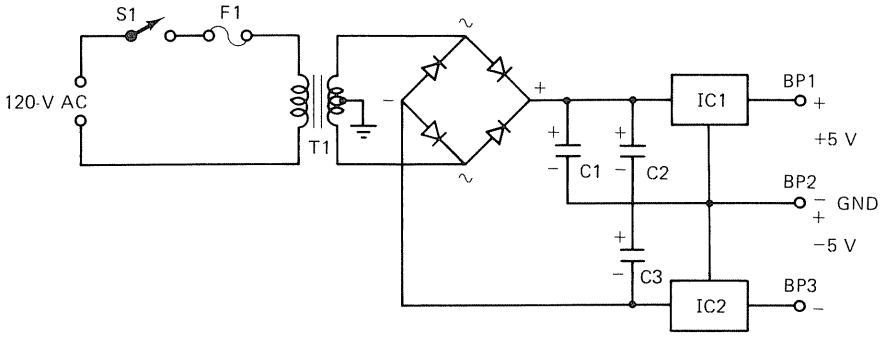



| Component | Radio Shack | Jameco |
|---|---|---|
| T1 | 9-V battery eliminator 270-1552 | DC 900 |
| C1 | 272-1020 2200 μF at 35 V | 2200 μF at 16 V |
| IC1 | 7805 276-1770 | LM340T-5 |

Figure I-2 Most experiments in this book require a +5-V power supply. This simple circuit uses a 9-V DC wall transformer, a 2200-μF filter capacitor, and a 7805 +5-V voltage regulator. Parts may be obtained at your local Radio Shack store or other parts house, such as Jameco Electronics (1355 Shoreway Road, Belmont, CA 94002). The connector on the wall transformer should be cut off and the wires connected to the capacitor and voltage regulator, which may be mounted on your breadboard. Be sure to observe the transformer output voltage polarity.

requiring only *three* components. The wall transformer (together with internal rectifier) converts the 120 V AC to approximately 9 V DC. The capacitor helps smooth out any voltage fluctuations, and the regulator integrated circuit maintains the output voltage constant at +5 V.

Figure I-3 illustrates another possible power supply circuit. This circuit




| Component | Radio Shack | Jameco |
|---|---|---|
| S1 on/off switch | SPST 275-612 | FTD01/JMT-123 |
| F1 fuse $\frac{1}{4}$ A | 270-1270 | AGC $\frac{1}{4}$ A |
| Fuseholder (panel mount) | 270-364 | HKP |
| T1 12.6-V CT transformer | 273-1505 1.2 A | – |
| Bridge rectifier 1A 50 PIV | 276-1161 | MDA 980-1 |
| C1, C2 2200 $\mu$F 35 V | 272-1020 | 2200 $\mu$F 35 V |
| C3 1000 $\mu$F 16 V | 272-1019 | 1000 $\mu$F 16 V |
| IC2 −5 V-regulator | – | LM320T-5 |
| BP1-3 binding posts | 274-662 | – |
| IC1 +5 V-regulator | 276-1770 7805 | LM340T-5 |

Figure I-3 This circuit provides both +5 V at 1 A and −5 V at 0.2 A output (the negative output is useful for the digital-to-analog and analog-to-digital experiments). It can be built on a small piece of perforated board and mounted in a suitable cabinet. Again, components are available from Radio Shack, Jameco Electronics, and most other electronic parts houses.

has the advantage of providing a −5-V output as well as +5 V. The negative output voltage is needed in the digital-to-analog and analog-to-digital experiments in Part 3.

## RIBBON CABLE ASSEMBLY

As mentioned earlier, a multiconductor ribbon cable will allow you to connect wires to the TRS-80 expansion port connector. This cable is a very critical part of the interface, and not just any cable can be used. If you have a Model I computer, the cable must have a *40*-pin *card edge connector* at one end (this end connects to the TRS-80) and a *socket connector* at the other. If you have a Model III computer, the cable must have a *50*-pin card edge connector at one end and a socket connector at the other. *Appendix A* lists the signals that are on the various pins of these connectors and discusses how to connect the cable properly to your computer.

Figure I-4 illustrates the 50-conductor cable and the solderless breadboard assembly. Note that the socket connector will allow No. 22 or 24 gauge wire connections between it and the breadboard. The breadboard

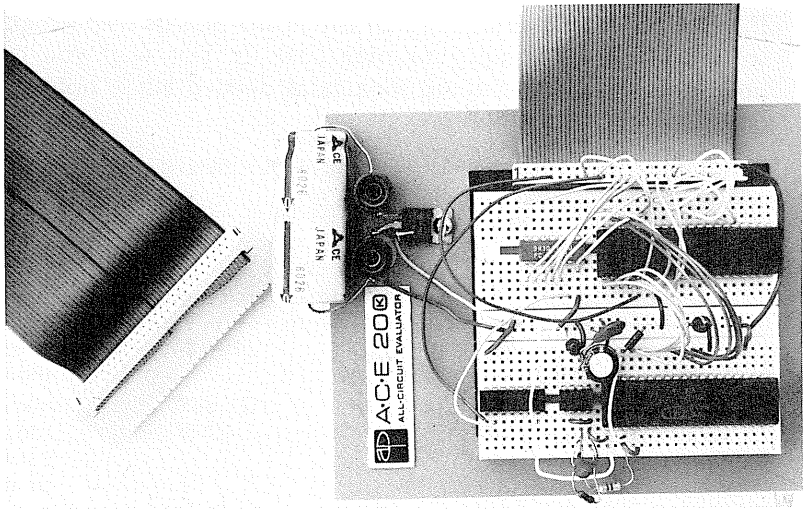



Figure I-4 This photograph illustrates the entire breadboard interface. The power supply (Fig. I-2) is mounted on the breadboard, with the base plate serving as a *heat sink* for the +5-V regulator. Banana plugs on the wall transformer cable facilitate connection to the breadboard. The two solderless breadboard sockets provide 68 rows of connections with a power bus (+5 V and ground) running between the two. The ribbon cable socket connector is held in place by the "stickum" on the base plate. The cable itself is a 50-conductor ribbon with a card edge connector on the computer end (a 40-conductor cable should be used with the Model I TRS-80). Refer to the text for sources of these components.

sockets have been offset slightly to the left to allow the socket connector to be held down by the *stickum* on the base plate.

Although it is possible to fabricate your own ribbon cable, a preassembled 40-conductor cable is available from Digi-Key Corp., P.O. Box 677, Thief River Falls, MN 56701. The part number is 924150-24. A 50-conductor cable is available from Priority One Electronics, 9161 Deering Avenue, Chatsworth, CA 91311. It is part number PRI50CESK.

## SOLDERLESS BREADBOARD

The purpose of the breadboard is to allow *easy* (solderless) connection between the various integrated circuits (ICs) used in the experiments and the TRS-80 computer. Referring to Fig. I-4, the ICs are placed so as to *straddle* the center divider of the socket. There are then four remaining connections to each IC lead in each row. Connections on one side of the center are electrically isolated from the other side.

Table I-1 compares several types of solderless breadboards. The more rows per side, the more ICs you will be able to interconnect and the more *expensive* the breadboard. The first two entries in the table provide sockets only, whereas the last three have the sockets mounted on a small base (see Fig. I-4). If the cost of some of the Proto-Boards seems high, you may want to consider fabricating your own base. In this way you can expand your breadboard with additional sockets as needed while holding the initial cost down.

TABLE I-1 SOLDERLESS BREADBOARDS ARE AVAILABLE IN A VARIETY OF CONFIGURATIONS AND PRICES

| Type | Number of rows per side | Part number | Manufacturer |
|---|---|---|---|
| Modular breadboard socket (may be expanded) | 47 | 276-174 | Radio Shack about $10 |
| Quick Test Socket and bus strip (may be expanded) | 59 | QT-59S, QT-59B | Jameco Electronics 1355 Shoreway Road Belmont, CA 94002 about $15 |
| Proto-Board with Quick Test Sockets | 68 | ACE-200K, 923333 | Digi-Key Corp. P.O. Box 677 Thief River Falls, MN 56701 about $20 |
| Proto-Board with Quick Test Sockets | 84 | ACE-201K, 923334 | Digi-Key about $30 |
| Proto-Board with Quick Test Sockets | 94 | PB-102 | Jameco about $35 |

## INTEGRATED CIRCUITS (ICS)

Each experiment will require that a number of ICs be inserted into the solderless breadboard. Number 22 or 24 gauge wire will then be used to connect the IC pins together and to the ribbon cable socket connector. At the beginning of each experiment all necessary parts to do that experiment will be specified. In addition, *Appendix B* lists all components needed to do the experiments in this book. You may want to order all of these parts or only those for the experiments you choose to do.

## NICE-TO-HAVE COMPONENTS

There are additional supplies that can make your journey through this book more meaningful and be an aid to you in troubleshooting your circuits.

## LOGIC PROBE

A digital circuit may have only two operating states: *ON* or *OFF*. The output of a logic gate is either high (1) or low (0). Although we could monitor this condition with a voltmeter, another tool is often used—a *logic probe*. Figure I-5 pictures a typical probe. It is powered by +5 V and is then touched to the various points in the circuit to be tested. Three light-emitting diodes (LEDs) generally serve as the logic-level indicators. These correspond to the

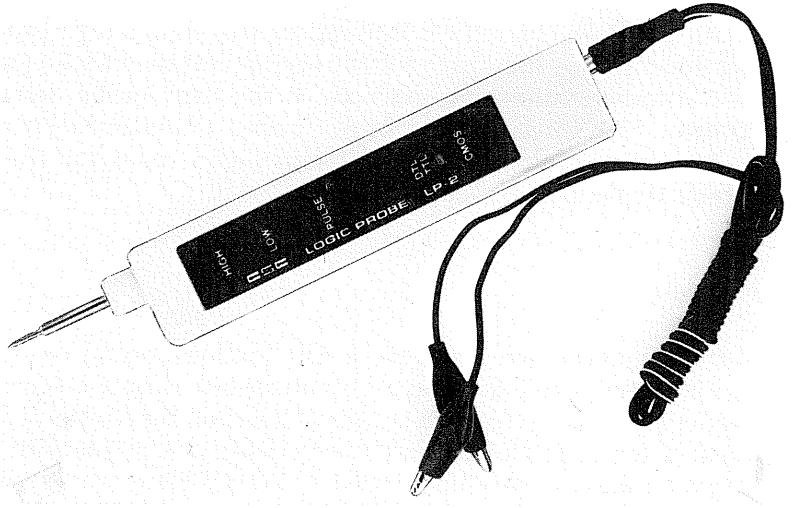



**Figure I-5** A logic probe will be handy for monitoring logic levels. Its three LEDs brightly indicate a logic 1, a logic 0, or a pulse condition.

logic 1 and 0 states and to a pulse condition (a logic-level switching rapidly between the 1 and 0 states).

A logic probe will be very handy for monitoring circuit conditions in the various experiments in this book. Often we will be programming the TRS-80 to output a certain binary pattern of 1's and 0's which can easily be monitored with the logic probe. Table I-2 lists several common logic probes, their distinguishing features, and suppliers.

TABLE I-2 VARIOUS COMMONLY AVAILABLE LOGIC PROBES[a]

| Source | Part number | Description |
|---|---|---|
| Radio Shack<br>about $25 | 22-300 | Fully assembled, detects pulses to 300 ns |
| Jameco Electronics<br>1355 Shoreway Road<br>Belmont, CA 94002<br>about $23 | LPK-1 | Logic probe kit, detects pulses to 300 ns; $f_{max}$ = 1.5 MHz |
| Jameco<br>about $50 | LP-1 | Fully assembled, detects pulses to 50 ns; $f_{max}$ = 10 MHz |

[a]A logic probe, although not essential, will be handy for monitoring logic levels in the interface circuits.

Again you can save money by building your own. There have been numerous articles on logic probe construction over the last few years—for example, Steve Dominguez, "Probos V," *Kilobaud Microcomputing*, October 1979, p. 78. In this article Dominguez describes a logic probe with capabilities equal to those in Table I-2 but costing less than $5 for parts!

Another interesting example is the logic probe described by Robert Kreiger, "Build an Audible Logic Probe," *Popular Electronics*, July 1980, p. 73. This circuit produces different audio tones for the three logic conditions (high, low, and pulse).

## DC VOLTMETER

Although again not essential, a DC voltmeter may be handy to measure power supply voltages and to troubleshoot simple electrical circuits. It is particularly handy to have an ohms function for continuity testing in cables (testing for *open* circuits and *short* circuits). A meter with volts, ohms, and current ranges is generally called a *VOM*. There are numerous sources for such meters, with prices ranging from $5 to several hundred dollars.

## OSCILLOSCOPE

A dual-channel oscilloscope would be very useful for troubleshooting and tracing pulses through the various experiments. With it you would be better able to appreciate the dynamic nature of the interfacing circuits. If you have one—great, be sure to use it. If not, don't worry, all of the experiments can easily be built and tested without one.

## CONCLUSION

Your next step should be to page through the experiments in Parts 2 and 3 and decide if you want to do all or just some of these experiments. Make yourself a *shopping list* (refer to Appendix B) and then prepare to enter the world of microcomputer interfacing.

# 2 Basic Concepts of Microcomputer Input/Output

The seven experiments in this section will teach you how to add input and output ports to the TRS-80. BASIC software for controlling these ports is also covered.

# EXPERIMENT 1

## THE FOUR I/O COMMANDS IN BASIC

### OVERVIEW

In this experiment you will wire a 7476 JK flip-flop as a latch to catch pulses generated on the *control bus* of the TRS-80. The **INP**, **OUT**, **PEEK**, and **POKE** commands will be used to generate these pulses.

### OBJECTIVES

The key points to be learned from this experiment are:

1. For interfacing purposes, the TRS-80 can be thought of as having three buses: the *data bus*, the *address bus*, and the *control bus*.
2. The control bus consists of the $\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{OUT}}$, and $\overline{\text{IN}}$ lines. (*Note:* The Model III provides only the $\overline{\text{OUT}}$ and $\overline{\text{IN}}$ lines.)
3. These lines briefly pulse low when the **PEEK**, **POKE**, **OUT**, and **INP** commands are executed.
4. These control pulses are active for less than 1 $\mu$s and, therefore, to be "seen" must be caught by a latch or monitored with a logic probe.

### PARTS LIST

1 7476 JK flip-flop
1 LED (light-emitting diode)
1 180-$\Omega$ resistor (brown-gray-brown)

## DISCUSSION

### Three-Bus System Architecture

A digital signal may exist in only one of two legitimate states. These are referred to as the *ON* and *OFF* or 1 and 0 states. In practice, these states usually correspond to different voltage levels. For example, for the 7400 TTL (transistor–transistor logic) family, a 0 is any voltage *less than* or equal to 0.4 V and a logic 1 is any voltage *greater than* or equal to 2.4 V. Typical voltages are 0.2 V and 3.4 V. In a CMOS (complementary metal-oxide semiconductor) logic gate, the typical levels are 5 V for a logic 1 and 0 V for a logic 0.

Other schemes do exist and require special interface circuits when communicating with TTL. For example, the ASR-33 teletype uses a *20-mA current loop*. The presence of current is a 1; the absence of current, a 0. Still another system, *RS-232C*, considers a 1 to be a voltage less than -3 V and a 0 to be a voltage greater than +3 V. As a final example, a *modem* allows digital communications over the telephone lines by converting the 1's and 0's to audio frequencies. Typically, 1270 Hz represents a 1 and 1070 Hz represents a 0.

All signals provided by the TRS-80 at its expansion port connector can be considered to switch between ~3.4 and 0 V. This means that all signals are TTL compatible.

When a number of digital signals or lines are used for a common purpose, these lines are referred to as a *bus*. Because all the lines connected to the expansion port connector are there for a common purpose (expansion of the computer), we might call this port the *expansion bus*. Actually, within this single bus (40 pins for the Model I and 50 pins for the Model III) there are several smaller buses. These include the three main buses we will be concerned with for interfacing purposes: the *data bus*, the *address bus*, and the *control bus*.

### TRS-80 System Block Diagram

Figure 1-1 illustrates the TRS-80 in block diagram form (actually it could represent *any* Z-80 microcomputer system). The main components of the system are illustrated. The clock generator establishes the basic timing for the entire computer. The Z-80 CPU (central processing unit) acts as a "traffic cop" directing the flow of data between the various system components as it is directed by programs stored in RAM (random access memory) and ROM (read-only memory). The latter two blocks can be thought of as *programmable* user memory and *permanent* operating system memory, respectively.

The keyboard and video screen are actually *memory-mapped* devices and can be considered part of the ROM and RAM blocks.

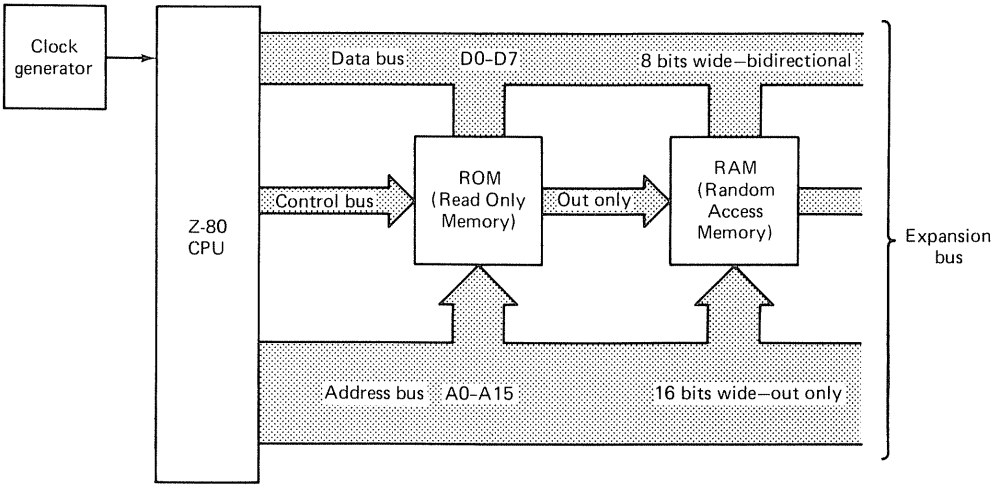



Figure 1-1 TRS-80 system block diagram. All components in the system interface to the Z-80 CPU over the data bus, address bus, and control bus. Refer to Appendix A and the text for specific details on signals brought out to the TRS-80 expansion bus.

Each of the three buses shown has a specific responsibility that we must become intimately familiar with.

1. *Data bus.* These eight lines handle the actual 8 bits of information (one byte) that the CPU is working with. These bits may represent *data* bytes or instruction *operation codes.*

2. *Address bus.* For the Z-80 (and most other microprocessors) the address bus consists of 16 output lines. The binary pattern placed on these lines by the CPU determines the specific memory location to be read from or written to. Because there are 16 address lines, there are $2^{16}$ or 65,536 different addresses possible, ranging from all 16 lines being low (address 0), to all 16 lines being high (address 65,535). The specific pattern on these lines is always generated by the CPU (except for direct memory access) and therefore these lines are outputs only.

3. *Control bus.* This bus consists of four lines called memory read ($\overline{RD}$), input read ($\overline{INP}$), memory write ($\overline{WR}$), and output write ($\overline{OUT}$). The bar above each identifies it as an *active low* signal. This means that the line is a logic 0 when activated. Note that Radio Shack uses RD*, INP*, WR*, and OUT* to represent these active low signals. These four control lines alert the outside world (outside the CPU, that is) to what the computer is about to do next. Basically, they identify if the current operation (machine cycle) will require data to *enter* the CPU (memory or input read) or *leave* the CPU (memory or output write).

The three-bus structure just described requires a total of 28 lines. On the Model I computer all 28 of these lines (plus 12 others) are brought out to the expansion connector. On the Model III computer, only 25 unique lines are brought out (25 of the connector's 50 pins are actually connected to ground). Appendix A details the specific pin numbers and descriptions of both computers' expansion connectors and will be referred to extensively in these experiments.

### Read and Write Operations

Figure 1-2 illustrates how the three buses are used for a *read* operation. A memory read is illustrated in Fig. 1-2a and an input read is illustrated in Fig. 1-2b. In both cases the CPU first issues the address of the memory location (Fig. 1-2a) or I/O port (Fig. 1-2b) to be read from, and then activates the memory read ($\overline{RD}$) or input read ($\overline{INP}$) control signal. This signal is in turn used by external hardware to gate data onto the data bus and into the CPU.

Figure 1-3 illustrates the corresponding waveforms for a memory write (Fig. 1-3a) and output write (Fig. 1-3b) operation. Again, the CPU first outputs an address on the address bus. This may be the address of a memory cell or an output port. The data to be written is then placed on the data bus and gated into memory by $\overline{WR}$ or into an output port by $\overline{OUT}$.

### Types of I/O

In general, a microprocessor may interface with its I/O devices in two ways. In the first, called *I/O-mapped I/O*, the microprocessor must use special INP and OUT instructions. In the case of the TRS-80, the I/O address is placed on the *low-order* byte of the address bus (A0 through A7) and data is input with a BASIC command such as

```
10  Y=INP(47)
```

or output with a command such as

```
10  OUT 47,Y
```

In the first case the variable Y receives the data from port 47, while in the second case, the variable Y is output to port 47.

Note that because only one byte is used for the port address (A0 through A7), there are only 256 ($2^8$) possible I/O port addresses.

A second type of I/O is called *memory-mapped I/O*. In this scheme I/O devices are interfaced so as to appear to be memory locations and the *full 16-bit* address bus is used. In this case, any command that reads or writes

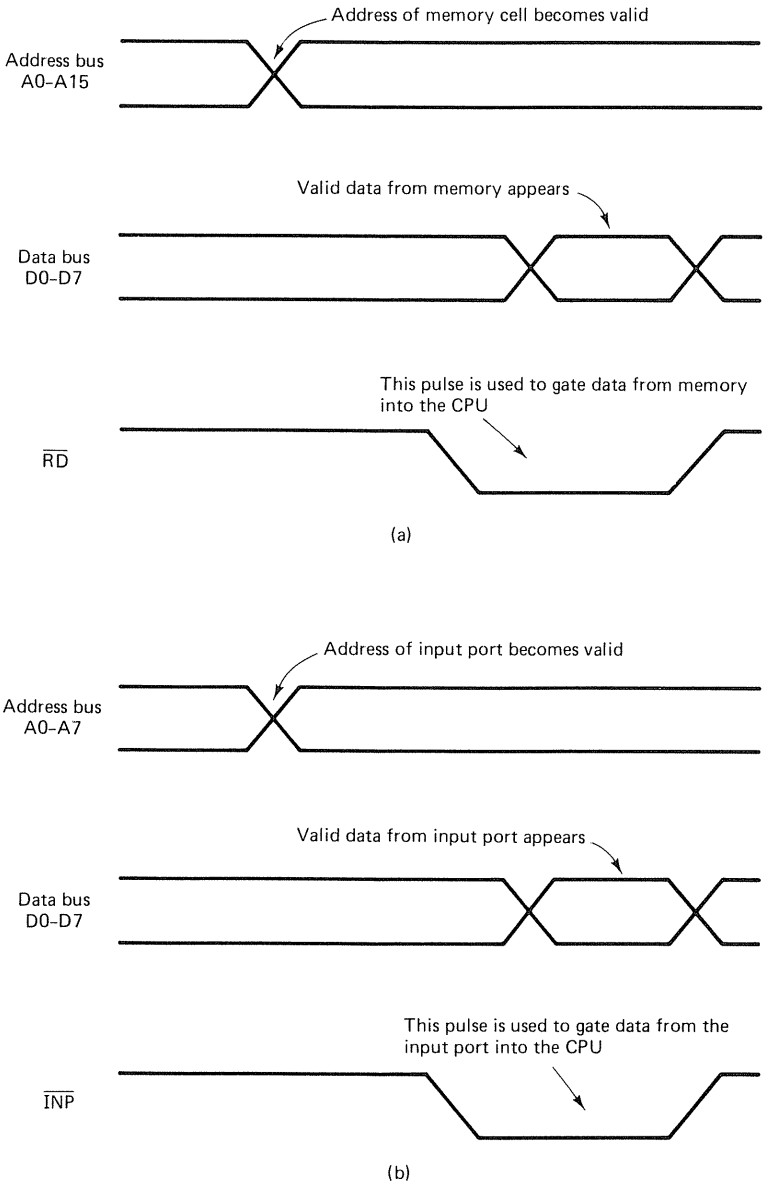



Figure 1-2 In (a) a memory read cycle is shown. In (b) the identical operation occurs except that data is read from an input port instead of from memory. In this case the $\overline{INP}$ line goes low instead of the $\overline{RD}$ line.

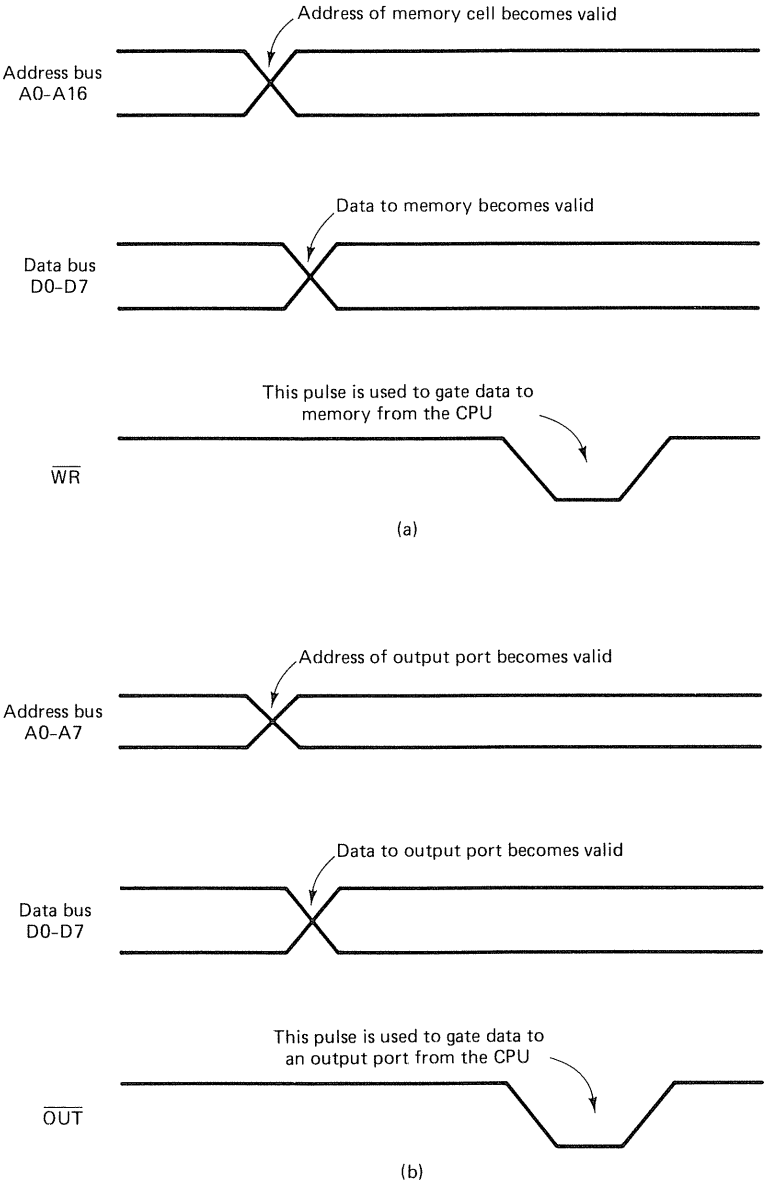



Figure 1-3 In (a) a memory write cycle is shown. In (b) the identical operation takes place except that data is written to an output port. In this case the $\overline{OUT}$ line goes low instead of the $\overline{WR}$ line.

to memory becomes an I/O command. In BASIC there are just two. For example, to read the contents of port 26128, we would use

```
10 Y=PEEK(26128)
```

Similarly, to write the contents of Y to this same port, we could use

```
10 POKE 26128,Y
```

In the experiments to follow, we explore both memory-mapped and I/O-mapped I/O interfaces. In these experiments the hardware differences will be illustrated.

*Note to Model III users.* Because the high-order address lines (A8 through A15) and $\overline{RD}$ and $\overline{WR}$ control lines are *not* brought out to the expansion connector of your computer, you will be unable to perform the memory-mapped I/O experiments.

## PROCEDURE

**Step 1.** Refer to Fig. 1-4a and wire this circuit on your breadboard. Be sure to observe the following:

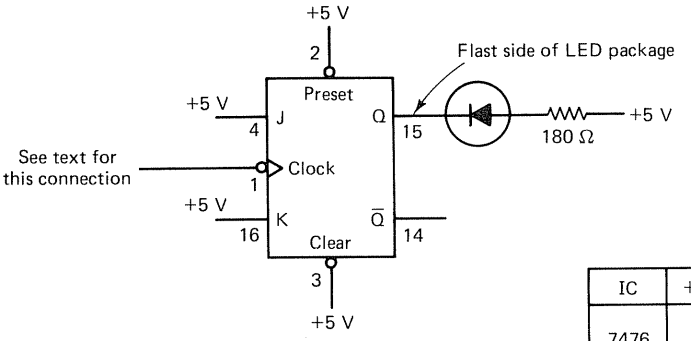
1. Position the IC so that it *straddles* the center notch (see Fig I-4a and 1-4b for an example).
2. The IC must have power: +5 V on pin 5, ground on pin 13.
3. Be sure to connect the flat side of the LED (the cathode) to pin 15 as shown in Fig. 1-4a.

**Step 2.** With all power *off*, connect your ribbon cable to the expansion port connector on your computer. Refer to Appendix A for the proper procedure to be used for your computer model.

**Step 3.** Referring to Appendix A for the connector pinouts of the expansion connector, locate the $\overline{OUT}$ pin number (pin 12 on the Model I and pin 35 on the Model III). Connect a wire from this pin on the *socket connector* to pin 1 of the 7476.

**Step 4.** Connect a wire between any pin on the breadboard ground bus to a ground pin on the TRS-80 expansion bus (for example, pin 8 on the Model I or pin 2 on the Model III).

**Step 5.** The 7476 flip-flop is wired in the *toggle* mode (see Appendix E for a review of basic flip-flop operation). Each time an $\overline{OUT}$ pulse occurs, the Q output will switch to its opposite state (toggle). We will observe this




| IC | +5 V | GND |
|---|---|---|
| 7476 | 5 | 13 |

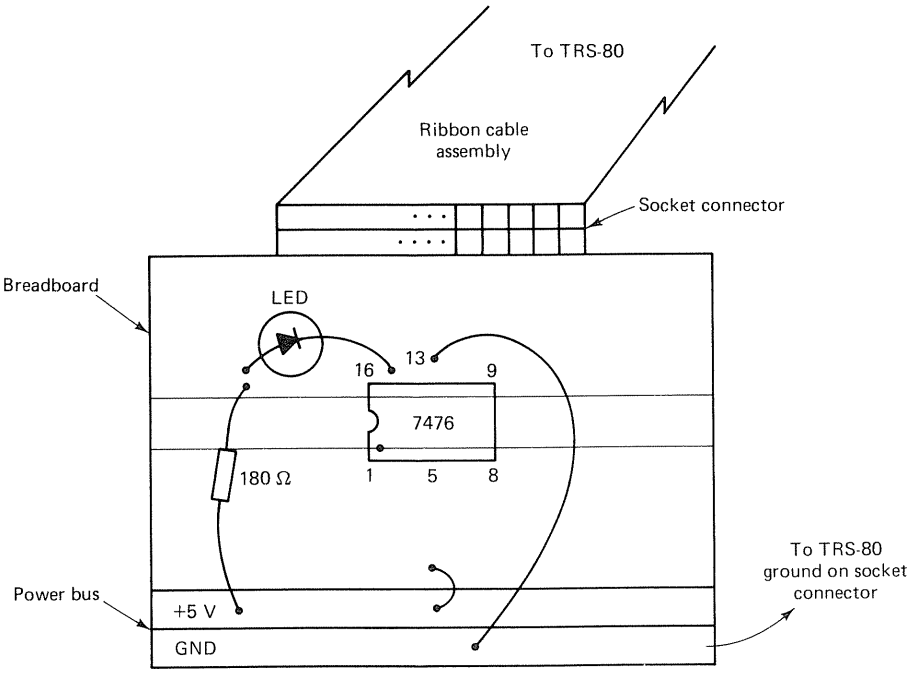



Figure 1-4 (a) Test circuit to monitor pulses on the control bus of the TRS-80. (b) Pictorial view of the breadboard and interface cable. The 7476 is shown straddling the center notch of the breadboard. Note that pin 1 of the IC is identified by the dot or notch in the package.

by the LED turning *ON* and *OFF*. Enter the BASIC program below into your TRS-80 and run the program.

```
10  OUT  236,16        :REM NEEDED FOR MODEL III ONLY
20  OUT  47,Y          :REM GENERATE OUT PULSE
30  GOTO 20            :REM KEEP LOOPING
```

**Question 1-1.** What do you observe to happen? Can you explain the result?*

**Step 6.** Modify the program in step 5 by adding a time delay such as

```
25  FOR J=1 TO 250    :NEXT J
```

The LED should now switch *ON* and *OFF* at an approximately 1-Hz rate.

**Question 1-2.** Do you think the output port address (47 in step 5) will affect the LED flashing rate? Try other port numbers to verify your answer.

**Step 7.** Modify the program in step 6 by changing line 20 to

```
20  Y=INP(47)          :REM GENERATE AN IN PULSE
```

Rerun the program.

**Question 1-3.** Why does the LED no longer blink? What must be done to the *hardware* to cause the LED to blink?

*Note.* The programs you have just run caused the $\overline{OUT}$ and $\overline{IN}$ control signals to be activated. Because these pulses are present for less than *1 μs*, they cannot be connected directly to an LED and observed. Instead, we used them to *toggle* a flip-flop at a rate dependent on the time delay in line 25 of the program.

*Note to Model III users.* The next three steps (steps 8 through 10) test the $\overline{RD}$ and $\overline{WR}$ control lines. Because these lines are unavailable on your computer, you will be unable to perform these steps.

**Step 8.** Change the hardware in Fig. 1-4 so that pin 1 of the flip-flop is connected to the $\overline{RD}$ pin on the ribbon cable (pin 15 on the Model I).

**Step 9.** Load the following program into your TRS-80 and run it observing the LED.

```
10  Y=PEEK(26128)                  :REM GENERATE RD PULSE
15  FOR J=1 TO 250:   NEXT J
20  GOTO 10
```

*Answers to all questions are provided at the end of each experiment.

**Question 1-4.** How many times do you think the $\overline{RD}$ line pulses low during one pass through this program? Do other instructions besides the PEEK(26128) cause an $\overline{RD}$ pulse?

**Step 10.** Connect pin 1 of the flip-flop to the $\overline{WR}$ line (pin 13 on the Model I) and rerun the program in step 9 but with line 10 changed to

```
10 POKE 26128,Y          :REM GENERATE WR PULSE
```

You should observe a result similar to step 9. The BASIC interpreter is causing many $\overline{WR}$ pulses to occur in addition to the POKE instruction.

**Step 11.** Reread the objectives as stated at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

*Note.* Leave this circuit on your breadboard, as it will be used in Experiment 2.

## SOLUTIONS TO QUESTIONS

1-1. The LED appears to be continually but dimly lit. This is because it switches *ON* and *OFF* so rapidly that the human eye cannot follow it.

1-2. No. The OUT 47,Y command is used to generate the $\overline{OUT}$ pulse only. It will do this regardless of the specific port address.

1-3. Connect pin 1 of the flip-flop to the $\overline{IN}$ pin (pin 19 on the Model I, pin 33 on the Model III).

1-4. The $\overline{RD}$ line pulses low many times as the program completes one loop. This is because in addition to the PEEK command the BASIC interpreter does many *read* operations in fetching op codes (machine language instructions) from memory and executing them.

# EXPERIMENT 2

## ADDRESS DECODING

### OVERVIEW

In this experiment you will wire an 8-bit I/O port *decoder* circuit. Using this decoder, input and output port enable circuits will be built. The effects of partial decoding will also be observed.

### OBJECTIVES

The key points to be learned from this experiment are:

1. The CPU issues a *16*-bit address on address lines A0 through A15 when referencing a memory location and an *8*-bit address on address lines A0 through A7 when referencing an I/O device.
2. An *address decoder* recognizes one unique address by outputting a pulse when that address is active.
3. For I/O ports it is not sufficient *only* to decode the address bus. The *control bus* signals must also be used and the coincidence of these two events used to generate the port enable pulse.
4. *Partial* decoding results when some of the address lines are not examined by the decoder. In this case the I/O ports will appear to occupy more than one port address.

### PARTS LIST

1 7476 JK flip-flop
1 7430 8-input NAND gate

1 7432 quad 2-input OR gate
1 7404 hex inverter
1 LED
1 180-Ω resistor (brown-gray-brown)

## DISCUSSION

### Hardware Address Concepts

The function of a typical microprocessor is to receive data from an input device, process this data, and output new data to an output device. It does this by communicating with memory for specific instructions on where to get the data, what to do with it, and where to output it. This description leads to the classical picture of a digital computer with central processing unit, memory, and input and output ports. This is illustrated in Fig. 2-1.

The actual electrical mechanisms needed to accomplish this are the *data bus*, the *address bus*, and the *control bus*. The data transfers between the various blocks in Fig. 2-1 occur over the *bidirectional data bus*. The CPU identifies the particular memory location or I/O device by outputting that memory cell's or port's unique address on the *address bus*. In addition, it identifies if the data is for an I/O port or a memory location and if it is entering or leaving the CPU (a read or write operation) by pulsing appropriate lines on the *control bus*.

The address bus for the TRS-80 is 16 bits wide (A0 through A15) and is available at the expansion port connector (the Model III provides only A0 through A7, see Appendix A for specific pin numbers for both computers). With 16 bits there are 65,536 ($2^{16}$) different addresses possible.

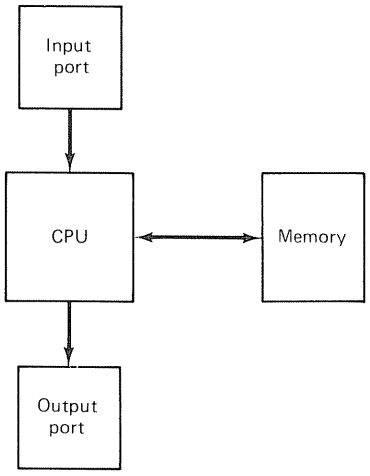



Figure 2-1 Block diagram of a typical digital computer. The CPU fetches instructions from memory, inputs data from its input ports, and outputs data to its output ports.

TABLE 2-1 THE FOUR I/O COMMANDS, THE CONTROL SIGNALS THEY ACTIVATE, AND THE RANGE OF ADDRESSES THEY CAN PUT ON THE ADDRESS BUS

| BASIC instructions | Control signal generated | Valid addresses |
|---|---|---|
| Y=PEEK(X) | $\overline{RD}$ | 0-65535[a] (A0-A15) |
| POKE X,Y | $\overline{WR}$ | 0-65535[a] (A0-A15) |
| OUT X,Y | $\overline{OUT}$ | 0-255 (A0-A7) |
| Y=INP(X) | $\overline{IN}$ | 0-255 (A0-A7) |

[a]The address must be computed if it exceeds 32767.

This is referred to as *64K* of memory space. In a typical microcomputer system the full 64K of memory is not usually implemented. For example, a 16K TRS-80 has 16K of RAM and 12K of ROM, leaving much of the memory space "open."

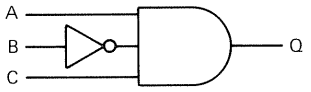
When addressing I/O devices, the TRS-80 uses only half of the address bus (A0 through A7) and is therefore limited to 256 ($2^8$) I/O ports.

We can control the specific address that is put on the bus and the control signal that is activated by using the four I/O commands discussed in Experiment 1. Table 2-1 summarizes this information. Note that the **PEEK** and **POKE** commands force the full 16-bit address to appear while the **INP** and **OUT** commands cause an 8-bit address to appear on A0 through A7.

## Address Decoding

A *decoder* is a circuit that recognizes one particular binary pattern or code. The simplest decoder possible is an *AND* gate (see Appendix D for a review of the various logic gate types). For example, a 3-input AND gate will have its output go high only when *all* three of its inputs are high. It recognizes the binary state 111.

A slight modification of this circuit allows it to decode *any* binary state. Figure 2-2 illustrates this concept. By selectively adding inverters we may decode any one of the eight possible states.




| C | B | A | Q |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

Figure 2-2 Simple decoder circuit. Only the input combination 101 will cause the output to go high.

| IC | +5 V | GND |
|---|---|---|
| 7404 | 14 | 7 |
| 7430 | 14 | 7 |

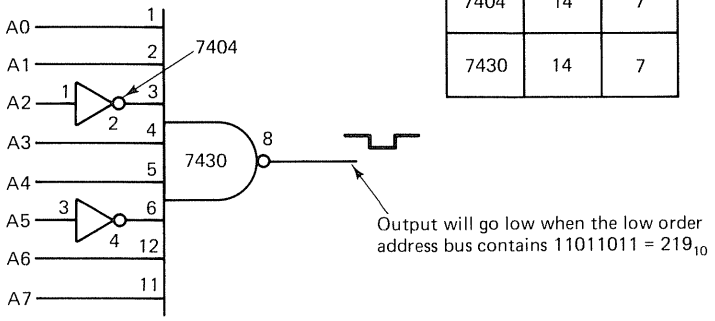



Figure 2-3 Active low I/O port decoder circuit. The output is normally high and goes low when the selected address occurs.

A practical port decoder circuit for the TRS-80 is shown in Fig. 2-3. This particular circuit uses a *NAND* gate, which means that the output will normally be high but will go *low* when the selected address occurs. This is illustrated by the low-going pulse on the 7430 output in Fig. 2-3. This type of output is referred to as *active low*, while the AND gate in Fig. 2-2 has an *active high* output.

More elaborate decoder schemes can be used. The circuit in Fig. 2-4

| IC | +5 V | GND |
|---|---|---|
| 7430 | 14 | 7 |
| 74LS138 | 16 | 8 |

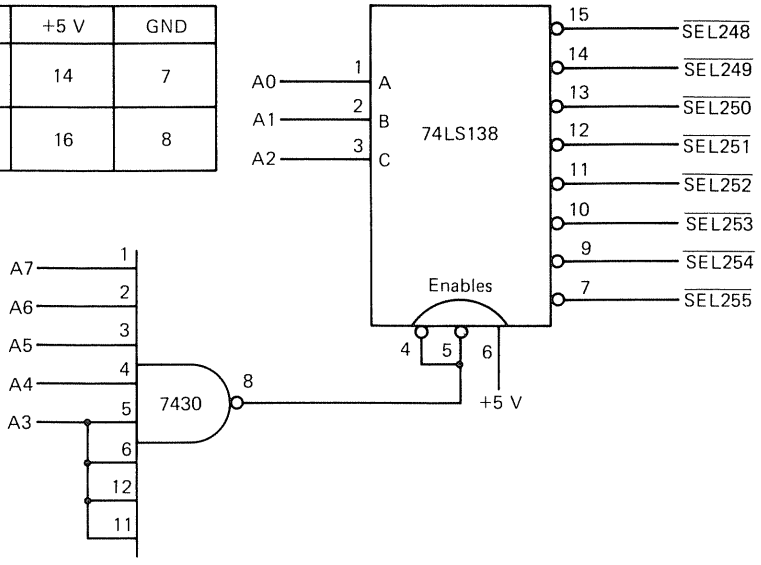



Figure 2-4 This circuit provides eight separate but consecutive address select pulses from decimal 248 (11111000) to decimal 255 (11111111). All outputs are active low.

uses a 74LS138 three-to-eight-line decoder. When the chip is enabled, it turns on one of its eight outputs corresponding to the binary code presented at the ABC inputs (A = least significant bit). In this particular schematic, the enables are wired so that A3 through A7 must be high in order that the 74LS138 be enabled. The eight possible combinations for A0, A1, and A2 then yield the eight outputs shown. This type of decoder is useful when a number of ports are to be interfaced to the computer.

**Example 2-1**

Design a port decoder to produce active low address select pulses for ports 0 through 7. Use a 74LS138.

**Solution** The circuit is shown in Fig. 2-5. Note that the alternate OR gate symbol has been used (see Appendix D for an explanation of this symbol). The 4075 is a CMOS triple three-input OR gate. One of the three gates is not used.

## Partial Decoding

When some of the address lines are not examined by the decoder, the resulting circuit is said to be partially decoded. An example of such a circuit is

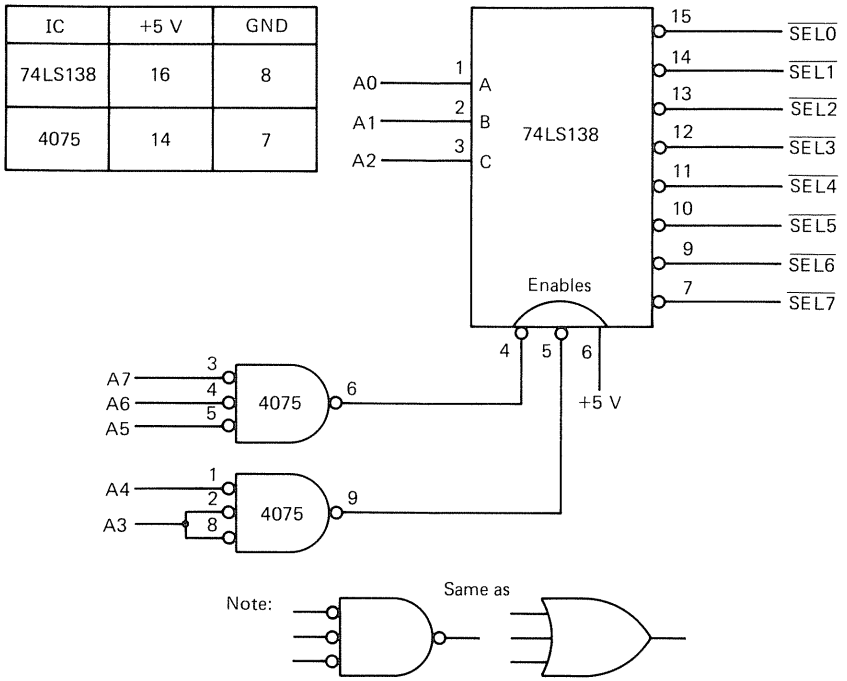
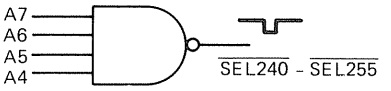



Figure 2-5 Solution to Example 2-1. Eight consecutive address selects are generated from port 0 through port 7.




| A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | x | x | x | x |

x = "don't care"

Figure 2-6 Example of a partial decoder. Because address lines A0 through A3 are not used by the decoder, they become "don't cares." The resulting circuit will then decode any address from decimal 240 through decimal 255.

shown in Fig. 2-6. The low 4 bits of the address bus are not used. This results in *16* different addresses (240 through 255) all activating the circuit as if they were the *same* address.

The main advantage to partial decoding is the resulting *simplification* in the hardware decoder. This is particularly true when decoding 16 bits. The main *disadvantage* is that one port now occupies the space of several ports, limiting the expansion of the system. Care must also be taken that two partially decoded ports do not overlap each other in their address assignments.

## I/O Port Enable

Input and output ports must receive an enable pulse telling them *when* they can latch the data on the data bus (an output port) or *when* to gate their data onto the data bus (an input port). The address select pulse alone is not sufficient to do this job. For example, when the CPU reads from memory, its low-order address (A0 through A7) could coincide with an output port address, causing the decoder's output to become active. But in this case we do not want to activate the output port and latch the data bus contents, because it contains data being read from memory. As another example, an input port and an output port may have the *same* address but share a single decoder. How does the CPU know which port to enable?

The answer to this question requires an understanding of the control bus signals. Refer to Table 2-1. When the CPU does an *input* instruction, it lets the outside world know this by activating its $\overline{IN}$ control line. Similarly, a *memory read* differs from an *output write* because the $\overline{RD}$ line will become active, not the $\overline{OUT}$ line (even if the low-order addresses are the same).

A circuit illustrating this point is shown in Fig. 2-7. When the low-order address bus contains $219_{10}$ AND the control bus signal $\overline{OUT}$ is active, the $\overline{OUT219}$ signal will go low. For the same reason, the $\overline{IN219}$ signal will go low if the control bus signal $\overline{IN}$ is active during this time. Because the processor will never make more than one of its control bus signals active at a given time, no conflicts can occur.

In the next two experiments we will see how these port enable pulses are used to interface input and output ports to the TRS-80.

| IC | +5 V | GND |
|---|---|---|
| 7430 | 14 | 7 |
| 7432 | 14 | 7 |
| 7404 | 14 | 7 |

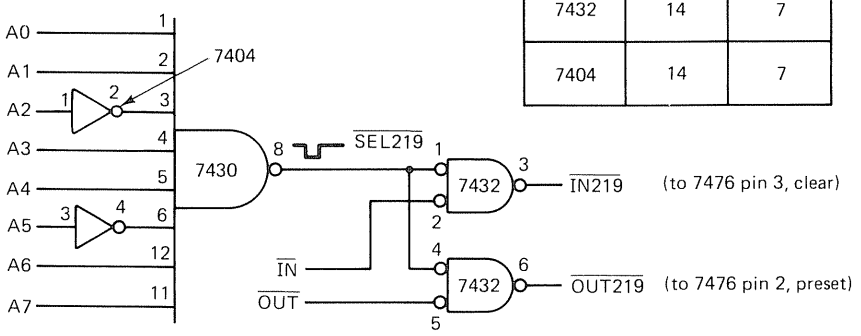



Figure 2-7 *Port select* pulses are generated by finding the coincidence of the address select pulse and the appropriate control bus line. In this example an input and output port select are generated sharing the same decoder circuit.

## PROCEDURE

**Step 1.** Refer to Fig. 2-8 and wire this circuit on your breadboard. (*Note:* Never wire a circuit with the breadboard or computer power on.) You will require three ICs to complete the wiring, although you may have the 7476 already wired from Experiment 1. *Be sure that you have a common ground wire between your breadboard and the TRS-80.*

| IC | +5 V | GND |
|---|---|---|
| 7404 | 14 | 7 |
| 7430 | 14 | 7 |
| 7476 | 5 | 13 |

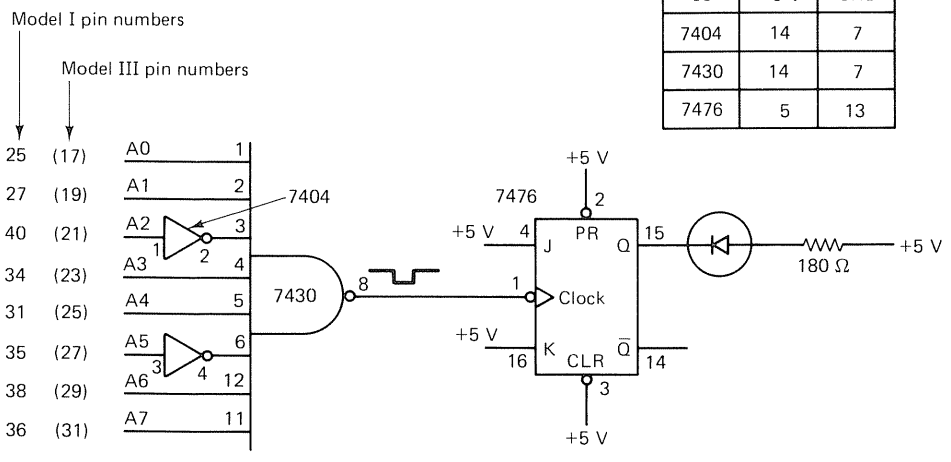



Figure 2-8 Circuit to be wired in step 1 of Experiment 2.

**Step 2.** The circuit you have just wired is similar to Fig. 2-3. Each time the *low-order* address bus contains $219_{10}$, the 7430 decoder output will pulse low and the flip-flop will receive a clock pulse. This will cause the Q output to switch states and the LED to go *ON* or *OFF*. Load the following program into your computer and run it observing the LED.

```
 5 OUT 236,16                  :REM MODEL III ONLY
10 OUT 219,Y                   :REM PUT 219 ON LOW-ORDER ADDRESS BUS
20 FOR J=1 TO 250: NEXT J:     :REM SLOW DOWN
30 GOTO 10
```

**Question 2-1.** You should have observed the LED to be lit. Why didn't it slowly switch *ON* and *OFF*?

**Step 3.** Modify the circuit in Fig. 2-8 so that the address decoder output is combined with $\overline{\text{OUT}}$ and *then* applied to the flip-flop, as shown in Fig. 2-9. Rerun the program from step 2 and again observe the LED.

**Question 2-2.** Explain why the LED now switches states only once for each cycle through the program.

**Step 4.** Change the port address in line 10 to some other number and run the program. Convince yourself that this circuit recognizes only port 219.

| | +5 V | GND |
|---|---|---|
| 7432 | 14 | 7 |

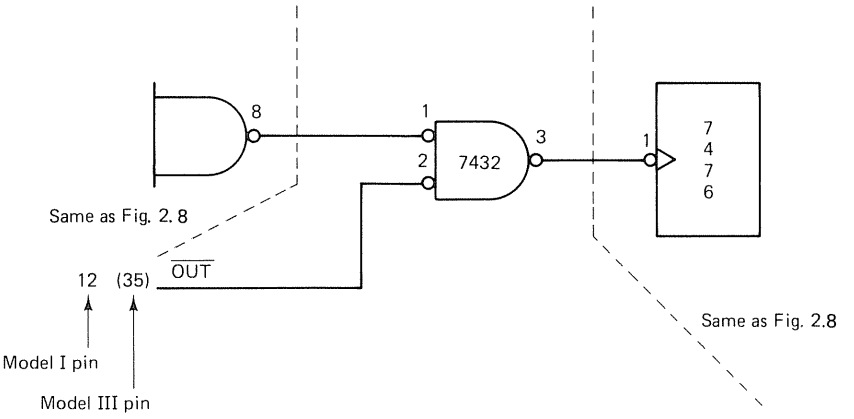



Figure 2-9 The circuit in Fig. 2-8 is changed by adding the 7432 OR gate between the decoder output and the flip-flop input.

**Step 5.** Change the program in step 2 line 10 to

```
10  Y=INP(219)
```

Rerun the program and observe the LED.

**Question 2-3.** What change to the *hardware* is now needed to cause the LED to blink? Try it.

**Step 6.** Rewire your circuit to agree with Fig. 2-7. Remove the +5-V connections from the flip-flop preset and clear inputs (pins 2 and 3). Wire $\overline{\text{OUT219}}$ to pin 2 and $\overline{\text{IN219}}$ to pin 3. Connect the flip-flop clock input (pin 1) to +5 V. Enter the following program into your computer and run it.

```
 5  OUT 236,16                          :REM MOD III ONLY
10  OUT 219,Y                           :REM TURN OFF LED
20  FOR J=1 TO 250:   NEXT J:           :REM KILL TIME
30  Y=INP(219)                          :REM TURN ON LED
40  FOR J=1 TO 250:   NEXT J:           :REM KILL TIME
50  GOTO 10
```

**Question 2-4.** Once you feel that you understand the operation of this program, write a new program that asks if you want the LED *ON* or *OFF*, and takes the appropriate action (remember that the LED goes *ON* when the Q output goes low). After completing this question you should be able to appreciate that the port enable pulses produced by this circuit could be used for purposes other than enabling I/O ports (for example, to turn on a relay, a valve, a motor, etc.).

**Step 7.** Remove the address bus connections A0 through A3 from the 7430 and connect the corresponding gate input pins (1, 2, 3, 4) to +5 V. The circuit is now *partially* decoded. Load the following program into your computer and run it.

```
10  CLS
20  INPUT "WHAT PORT NUMBER WOULD YOU LIKE ";P
25  OUT 236,16                          :REM MOD III ONLY
30  FOR J=1 TO 4
40  OUT P,Y                             :REM TURN OFF LED
50  FOR I=1 TO 100:   NEXT I
60  Y=INP(P)                            :REM TURN ON LED
70  FOR I=1 TO 100:   NEXT I
80  NEXT J
90  GOTO 10
```

**Question 2-5.** What *range* of addresses did you find that would cause the LED to switch *ON* and *OFF*?

**Step 8.** Reread the objectives as stated at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

## SOLUTIONS TO QUESTIONS

2-1. The BASIC interpreter caused $219_{10}$ to appear on the address bus at times *other* than that due to the **OUT 219,Y** in line 10.

2-2. The only time the address bus contains $219_{10}$ and $\overline{OUT}$ is low occurs in line 10 of the program, when the command **OUT 219,Y** is executed.

2-3. Change the $\overline{OUT}$ connection to $\overline{IN}$ (Model I, pin 19; Model III, pin 33).

2-4. One possible solution is:

```
10 CLS
20 INPUT "DO YOU WANT TO TURN THE LED ON OR OFF ";A$
25 OUT 236,16                    :REM MODEL III ONLY
30 IF A$="OFF" THEN 70
40 IF A$ <> "ON" THEN 20
50 Y=INP(219)                    :REM TURN LED ON
60 GOTO 10
70 OUT 219,Y                     :REM TURN LED OFF
80 GOTO 10
```

2-5. You should have found that any address from 208 through 223 was decoded by the decoder and activated the flip-flop. The binary address is 1101XXXX.

# EXPERIMENT 3

## I/O-MAPPED OUTPUT PORT CONCEPTS

### OVERVIEW

In this experiment you will wire a 7485 4-bit comparator and four-position DIP (dual-in-line package) switch as a port decoder that is switch selectable to 1 of 16 addresses. You will use this circuit to enable an output port constructed from a 74100 8-bit latch.

### OBJECTIVES

The key points to be learned from this experiment are:

1. A computer output port requires an address *decoder*, a port *enable* circuit, and a *latch* to catch the data.
2. A latch is an essential component in an output port because the output data is on the data bus only for *fractions* of a microsecond.
3. The OUT X,Y command is used to force data Y onto the data bus and address X onto the low-order address bus.

### PARTS LIST

| | |
|---|---|
| 1 | 8-position DIP switch (four positions unused) |
| 8 | 180-Ω resistors (brown-gray-brown) |
| 8 | LEDs |
| 1 | 7485 4-bit comparator |
| 1 | 7430 8-input NAND gate |

1 7404 hex inverter
1 7432 quad OR gate
1 74100 8-bit latch

## DISCUSSION

### The OUT X,Y Command

When using BASIC, data is output to a port with the command **OUT X,Y**. The sequence of events is as follows:

1. The address of port X is placed on the *low-order* address bus (A0 through A7).
2. The data Y is placed on the bidirectional data bus (now configured as output lines).
3. The $\overline{\text{OUT}}$ control line goes low for approximately 0.5 $\mu$s, indicating that a valid address is now on A0 through A7 and output data is available on D0 through D7.

Because X and Y are limited to 8 bits each, their values are restricted to $255_{10}$ or less. You should be able to see that using the **OUT X,Y** command, we can force any data and address we choose to appear on the data and address buses.

The job the hardware must perform should then be clear. It must:

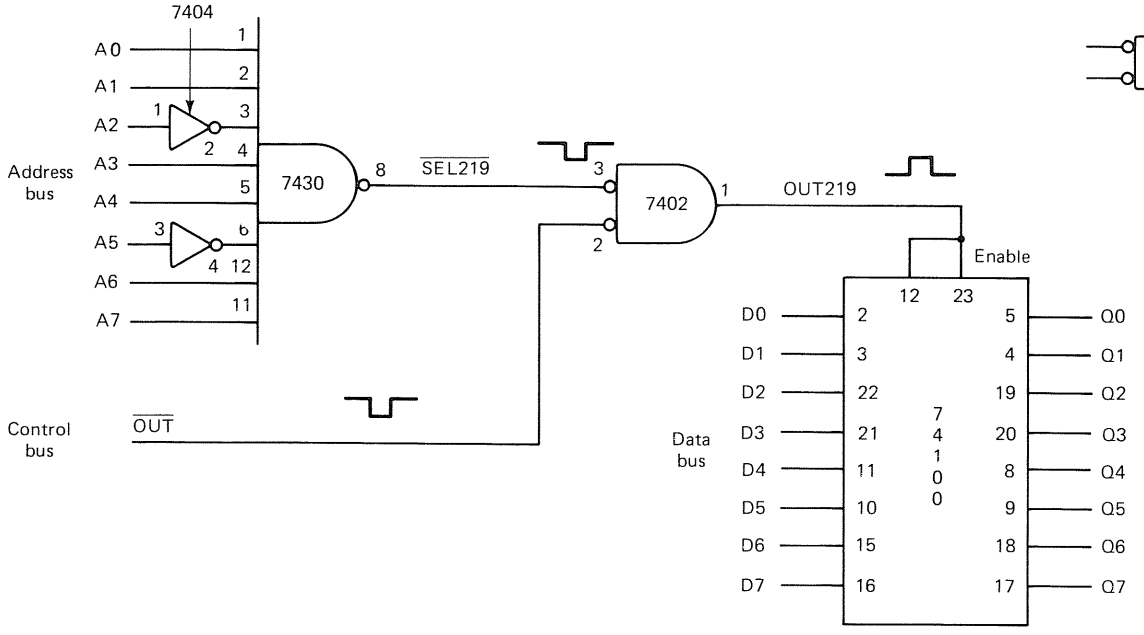
1. Decode the port address.
2. Use the address select pulse and $\overline{\text{OUT}}$ pulse to generate a *port enable* pulse.
3. Use this port enable pulse to enable a latch to store the output data.

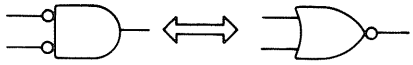
### Typical Output Port

Figure 3-1 illustrates schematically a typical microcomputer 8-bit output port. The 7430 functions as an *address decoder* and "looks" for address $219_{10}$ (11011011), which causes its output to go low. When this occurs AND the $\overline{\text{OUT}}$ control line goes low, the 7402 output will go high. This output port enable pulse (**OUT219**) is used to enable the 74100 8-bit latch (eight flip-flops). The latch stores whatever is on the data bus at this instant and it then becomes available for inspection at Q0 through Q7.

Note that what makes this circuit work is the *unique* conditions that exist on the three buses when the **OUT 219,Y** command is executed. *No other BASIC command can activate this circuit.*

Although this circuit uses a 74100 latch, other equivalent circuits can also be used. Table 3-1 is a partial list of several common latches together



| IC | +5 V | GND |
|---|---|---|
| 7404 | 14 | 7 |
| 7430 | 14 | 7 |
| 7402 | 14 | 7 |
| 74100 | 24 | 7 |

Figure 3-1 The port enable pulse (**OUT219**) generated by this circuit is used to enable a 74100 8-bit latch connected to the TRS-80 data bus. Note that this circuit is similar to Fig. 2-7 except that an active high enable pulse is required.

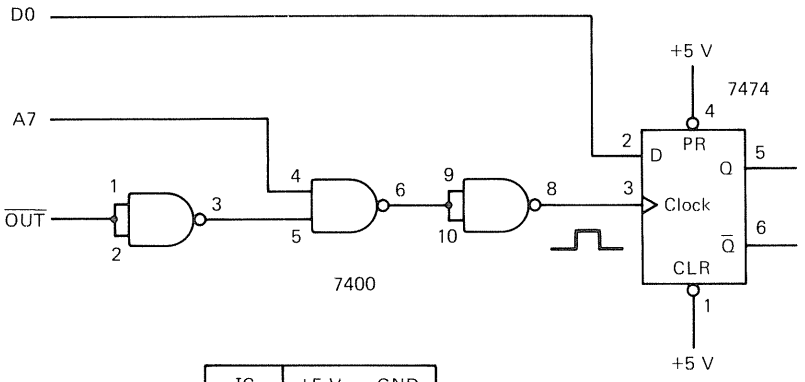
TABLE 3-1 TYPICAL LATCHES SUITABLE AS MICROCOMPUTER OUTPUT PORTS

| Part number | Description | Clock | Package type |
|---|---|---|---|
| 7475 | 4-bit latch | Level triggered | 16-pin DIP |
| 74100 | 8-bit latch | Level triggered | 24-pin DIP |
| 74173 | 4-bit latch with tri-state outputs | ↑ Edge triggered | 16-pin DIP |
| 74174 | 6-bit latch | ↑ Edge triggered | 16-pin DIP |
| 74175 | 4-bit latch | ↑ Edge triggered | 16-pin DIP |
| 74LS373 | 8-bit latch with tri-state outputs | Level triggered | 20-pin DIP |

with their distinguishing features. Latches such as the 7475 and 74100 are *level* triggered, meaning that their outputs will latch the input data when the clock is at a certain level (0 V in this case). The other latches in the table are *edge* triggered, meaning that the input data is latched at the instant the clock edge occurs (the rising edge in this case).

### The Data Output Is Always 8 Bits

Although this might already be obvious to you, there is no way to instruct the microprocessor to output less than 8 bits of data. Of course, we do not have to *latch* all 8 bits. For example, a very simple 1-bit output port is illustrated in Fig. 3-2. This circuit uses partial decoding, and any time that A7




| IC | +5 V | GND |
|---|---|---|
| 7400 | 14 | 7 |
| 7474 | 14 | 7 |

Figure 3-2 A 1-bit partially decoded output port. This circuit will respond to any port address $128_{10}$ or above and latch data bus bit D0.

is high AND $\overline{OUT}$ is low, the D flip-flop will receive a clock pulse and latch bit D0 of the data bus.

**Example 3-1**

Determine BASIC commands to *set* and *reset* the flip-flop in Fig. 3-2.

**Solution** Any address with A7 high will work: that is,

$$1XXXXXXX \quad \text{or} \quad 128_{10} \text{ to } 255_{10}$$

Then to set the flip-flop use

**OUT 128,1**

and to reset the flip-flop use

**OUT 128,0**

Suppose that the *8*-bit output port in Fig. 3-1 is wired on your breadboard and the command **OUT 219,73** is executed. What will we see at Q0 through Q7?

The answer, of course, is that we will see the binary equivalent of 73 or 01001001. Actually, the numeric value of the data output may not be as important as the *binary pattern* of 1's and 0's that it establishes. For example, Fig. 3-3 illustrates the 8-bit latch of Fig. 3-1 connected to various conveniences in the home. Let us assume that we wish to turn on the yard light, the bathroom light, the coffee pot, and the furnace but to keep the other devices off. What number should we output to port 219?

First, write down the desired binary pattern (assuming that a logic 1 will activate the device):

01011010

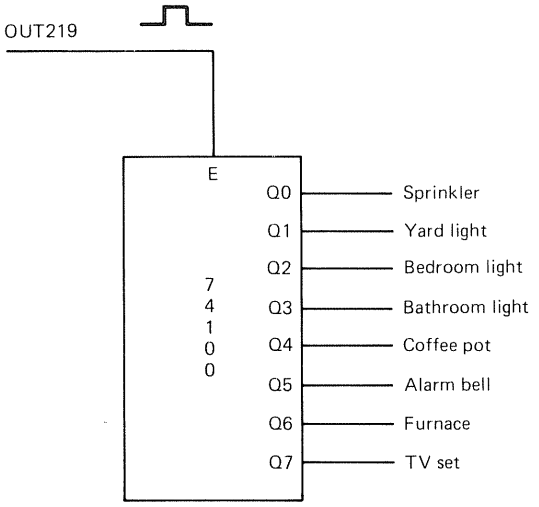



Figure 3-3 If desired, the 8 bits of a typical output port can be used to control appliances within the home. In this case, a particular *binary pattern* must be output to activate the proper appliances.

Now converting to decimal, $64 + 16 + 8 + 2 = 90_{10}$. The BASIC command would be **OUT 219,90.**

### Seven-Segment Display Output Port

Suppose that we wish to build an output port on our computer that can be used to count events in decimal from 0 to 99. A possible circuit is shown in Fig. 3-4. In this circuit we have arbitrarily selected output port 3 and have chosen two *common-anode* seven-segment displays. The 7475s will latch the data bus contents when A0 through A7 contain 00000011 AND the $\overline{\text{OUT}}$ line is low. Each 7475 output is then *decoded* to provide the proper seven-segment format for the two DL707 displays

Thinking that all is well, we might proceed to test our circuit by executing the command **OUT 3,5.** The displays show *05*. Now we try **OUT 3,52.** The displays show *34*! What is wrong?

What we have forgotten is that although BASIC "thinks" in decimal, when it outputs data to a port, it does so in *binary*. When $52_{10}$ is output, the actual pattern on the data bus is 00110100. The most significant display receives 0011 (3) and the least significant display receives 0100 (4). Hence we see 34 and *not* 52.

What can be done about this problem? It must be solved in *software*. We must adjust the decimal number before output so that it appears correctly in the displays. Because each display receives 4 bits, there will not be a carry into the most significant display until a number greater than *15* (1111) is output. The least significant display actually counts *units* (as in decimal), but the most significant display counts *16's* instead of 10's. What we must do is to convert the most significant digit to its decimal value assuming a weight of 16 for its position.

**Example 3-2**

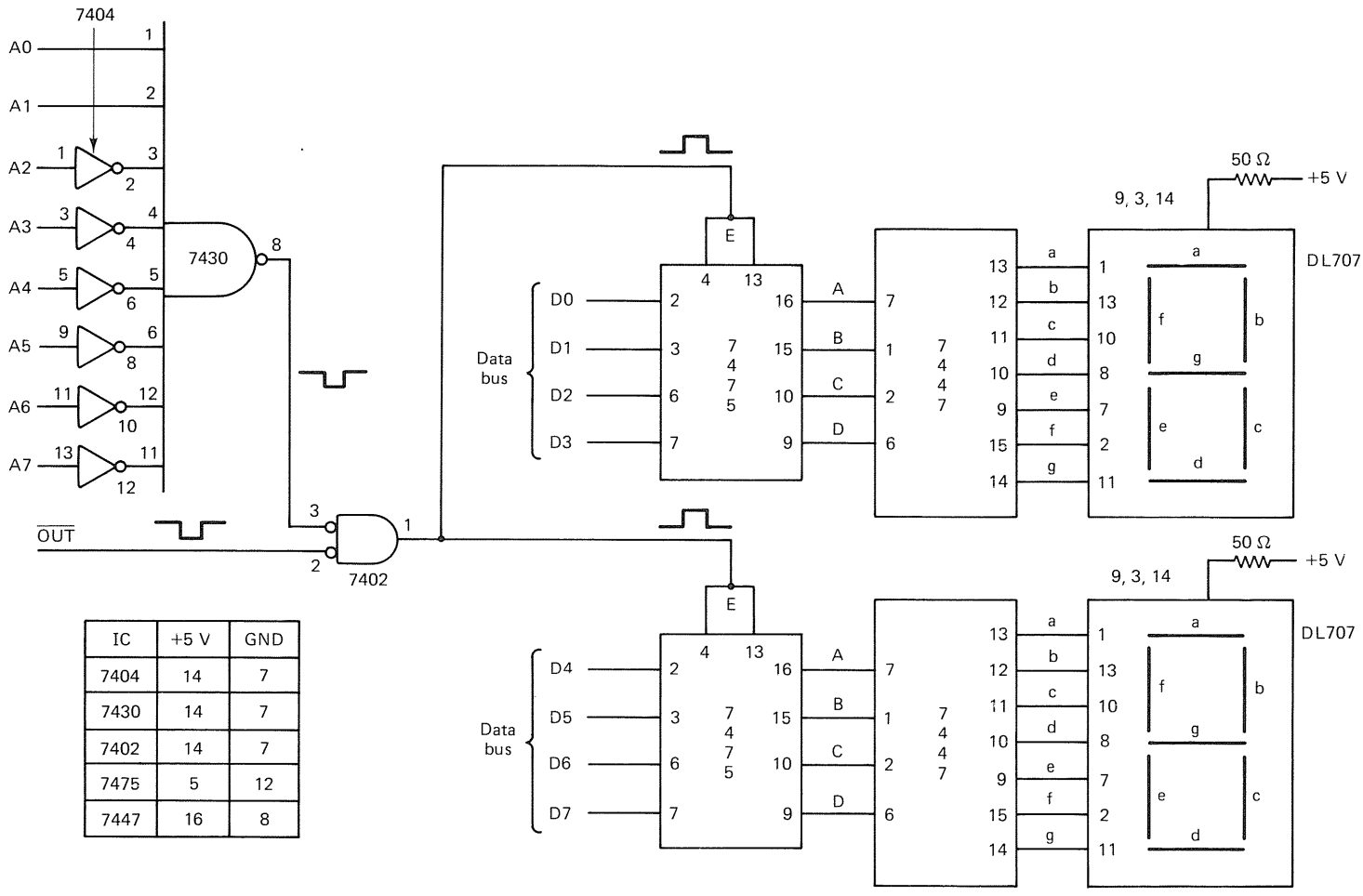
Convert $52_{10}$ to the proper format for output to two seven-segment displays.

**Solution**

1. The least significant display should receive the units digit without change, in this case 2.
2. The most significant display should show a 5. Converting to a base 16 for this digit, we have

$$5 \times 16 = 80$$

3. Add the two numbers: $80 + 2 = 82$.
4. Check: $82_{10}$ = 01010010 or 52 in the seven-segment displays.




| IC | +5 V | GND |
|---|---|---|
| 7404 | 14 | 7 |
| 7430 | 14 | 7 |
| 7402 | 14 | 7 |
| 7475 | 5 | 12 |
| 7447 | 16 | 8 |

Figure 3-4 Seven-segment display wired to output port 3.

A BASIC program that will properly output to the two seven-segment displays in Fig. 3-4 is listed below. Note that numbers greater that 99 cannot be displayed and are not allowed.

```
10 CLS
20 INPUT "ENTER ANY WHOLE NUMBER FROM 0 TO 99 ";N
30 IF N>99 OR N<0 THEN 20
40 N1=N-(INT(N/10)*10)
45 REM N1=LEAST SIGNIFICANT DISPLAY
50 N2=INT(N/10)*16
55 REM N2=MOST SIGNIFICANT DISPLAY
60 OUT 236,16        :REM MODEL III ONLY
70 OUT 3,N1+N2       :REM SHOW RESULT IN DISPLAYS
80 GOTO 20
```

As a final note, those of you who are up on your *hexadecimal* will note that the circuit in Fig. 3-4 actually displays the hexadecimal equivalent of the decimal number output. For example, when $52_{10}$ is output, the display shows 34, which is its *hexadecimal* equivalent.

Now, if you are also up on your seven-segment decoders, you will know that the 7447 will *not* display the hex characters A through F. If you are interested in building a hexadecimal display for your computer, replace the two 7475's and 7447's with *9368*'s. This chip provides a *4-bit latch* and a *hexadecimal decoder* in one package. You will need to use *common-cathode* displays, however, because the 9368 outputs are active high.

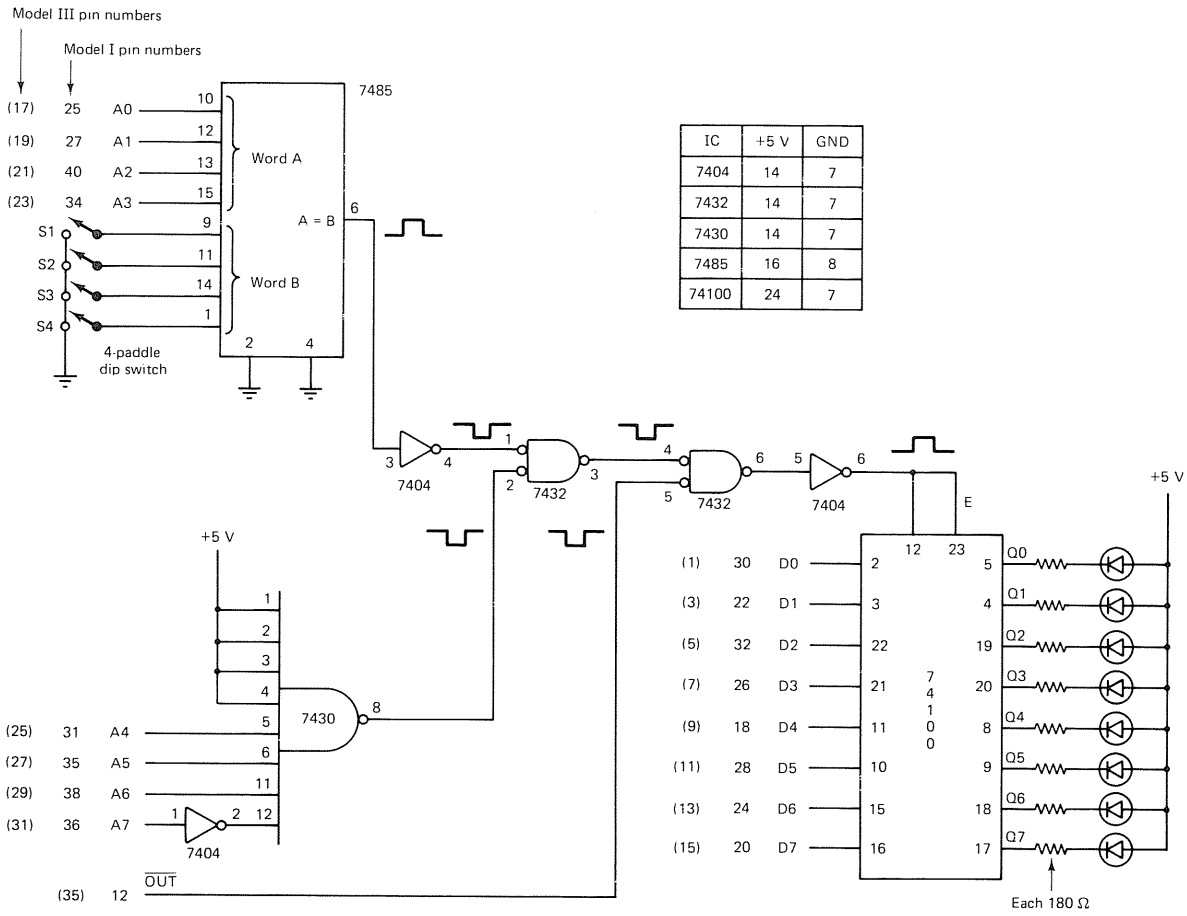
If you have a little more money to spend and wish to reduce the amount of wiring, use a *TIL311* or *HP 5082-7340* display. These chips have the latch, hexadecimal decoder, and LEDs all in one package. All that is needed is the power connections and the four binary inputs.

Refer to Experiment 12, Question 12-5, for a *software*-only technique for driving a seven-segment display. This method does away with the 7447 decoder chips.

## PROCEDURE

**Step 1.** Study the circuit in Fig. 3-5. Now carefully wire it on your breadboard noting the following:

1. Mount the DIP switch *upside down* on the breadboard with switch 1 to your right. In this way, when the switch handle is down it will correspond to a 0, and when up, to a 1.
2. Be sure that a *common ground* exists between the TRS-80 and your breadboard.




| IC | +5 V | GND |
|---|---|---|
| 7404 | 14 | 7 |
| 7432 | 14 | 7 |
| 7430 | 14 | 7 |
| 7485 | 16 | 8 |
| 74100 | 24 | 7 |

Figure 3-5 Schematic diagram of an 8-bit output port switch selectable to one of 16 different port addresses.

3. Be careful to orient all LEDs as shown (flat side of the package to the resistor).

*Note.* The 7485 is a 4-bit *magnitude comparator*. When word A (refer to Fig. 3-5) equals word B, its A = B output will go high. For this to happen, the switch settings (S1 through S4) will have to be *matched* by A0 through A3 of the address bus. In addition, A4 through A6 must be high and A7 low to enable the 7430. When both of these conditions are met AND $\overline{OUT}$ is low, the 74100 will be enabled. The range of port addresses is 01110000 to 01111111 or $112_{10}$ to $127_{10}$.

**Step 2.** Set the DIP switch to all 0's. Test your hardware with *one* of the following in the *immediate mode*.

```
OUT 112,0                    :REM MODEL I ONLY
```

or

```
OUT 236,16:   OUT 112,0      :REM MODEL III ONLY
```

**Question 3-1.** What should you observe? How do you turn all the LEDs *OFF*?

**Step 3.** Set all switches to the 1 (up) position. Again try outputting to port 112 and turn the LEDs ON or OFF. You should not be successful because the port code is no longer 112. Use the proper port address in the OUT command and try again. You should now be able to turn the LEDs *ON* and *OFF*.

**Question 3-2.** Write a BASIC program that asks you for the port address and the specific LEDs you want turned on. Run the program and verify proper operation. Remember that an *ON* LED requires a 0 output. Do not forget to set the switches to the port value you have chosen.

**Step 4.** Load and run the following program.

```
10 OUT 236,16                    :REM MODEL III ONLY
20 FOR J=255 TO 0 STEP -1
30 OUT 112,J
40 FOR I=1 TO 100:   NEXT I
50 NEXT J
60 GOTO 20
```

**Question 3-3.** What happens if line 20 is changed to the following?

```
20 FOR J=0 TO 255
```

**Step 5.** Write a BASIC program similar to the seven-segment display program at the end of the "Discussion" section. This program should input a

number between 0 and 99 and output to the LEDs. You will have to interpret the LEDs as two groups of four (Q0 through Q3 and Q4 through Q7) to verify program operation.

*Note.* Because a logic 1 will turn an LED *OFF*, you will actually see the complement (opposite) of what is desired. This may be fixed by changing your output command to OUT 112,255-(N1+N2).

**Step 6.** Reread the objectives as stated at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

## SOLUTIONS TO QUESTIONS

3-1. 0 = 00000000; all LEDs are *ON*. The command **OUT 112,255** will turn all LEDs *OFF*.

3-2. One possible solution:

```
 10 CLS
 15 L=0                :REM L IS DATA TO BE OUTPUT
 20 INPUT "ENTER THE PORT ADDRESS (112-127) ";P
 30 IF P>127 OR P<112 THEN 20
 40 FOR J=0 TO 7
 50 PRINT "TURN LED ";J; " ON/OFF ";
 60 INPUT L$
 70 IF L$="OFF" THEN L=L+(2↑J) ELSE IF L$<> "ON" THEN 50
 80 NEXT J
 85 OUT 236,16         :REM MODEL III ONLY
 90 OUT P,L
100 GOTO 10
```

3-3. Because a 0 output turns *ON* a LED, the LEDs will count backward when line 20 is changed.

*Note.* You may want to try other counting programs to cause various effects in the LEDs (for example, one LED appearing to bounce up and down).

# EXPERIMENT 4

## I/O-MAPPED INPUT PORT CONCEPTS

### OVERVIEW

In this experiment you will wire an eight-position DIP switch input port to the TRS-80 using a 74LS244 octal tri-state buffer. Software "masking" techniques will be demonstrated that allow individual switches to be monitored.

### OBJECTIVES

The key points to be learned from this experiment are:

1. Data from an input port must be gated onto the bidirectional data bus through *tri-state* gates.
2. The enable pulse for the tri-state gates is generated by the coincidence of the address select pulse AND the $\overline{IN}$ control signal.
3. The command **Y=INP(X)** is used to gate input data onto the data bus from port X and to store it as variable Y.

### PARTS LIST

| | |
|---|---|
| 1 | 8-position DIP switch |
| 1 | 74LS244 octal buffer |
| 1 | 7430 8-input NAND gate |
| 1 | 7432 quad 2-input OR gate |
| 1 | 7404 hex inverter |
| 8 | 1-kΩ resistors (brown-black-red) |

## DISCUSSION

### The Y=INP(X) Command

Data is input to the TRS-80 from BASIC with the command **Y=INP(X)**. When this command is executed, the following sequence of events takes place:

1. The address of port X is placed on the *low-order* (A0 through A7) address lines.
2. The bidirectional data bus configures itself for data *input*.
3. The $\overline{IN}$ control line pulses low for approximately 0.5 $\mu$s, indicating that the data bus is ready to receive data.

*Note to Model III users.* On your computer the data bus direction at the expansion port is controlled by pin 43, $\overline{EXT\ I/O\ SEL}$. This line must be driven low whenever an INP command occurs. This is easily done by connecting $\overline{IN}$ (pin 33) to $\overline{EXT\ I/O\ SEL}$ (pin 43). Refer to Appendix A for more details on this subject.

As was true with the output port, the input port address X and data input Y are limited to 8 bits each, restricting the port value and data input to *integers* between *0* and *255*.

In order to input data to the microcomputer, our hardware must:

1. Decode the port address.
2. Use the address select pulse and $\overline{IN}$ pulse to generate an input port enable pulse.
3. Use this enable pulse to gate data onto the data bus lines through *tri-state* gates.

You might have thought that we could connect the data *directly* to the data bus lines. The problem with this is that the input data would try to hold the data bus lines at specific logic levels, interfering with normal computer operation.

Figure 4-1 illustrates how the tri-state gate solves this problem. Only when the gate is enabled is the input logic level passed through the gate onto the bus. In fact, tri-state gates allow many *different* input ports to be connected to the *same* data bus. Because each set of tri-states is enabled by a separate address, there are no conflicts.

Tri-state gates are also used to increase the output drive capabilities of the microprocessor. If this is not done, the many circuits connected to the address and data bus may severely affect the processor's ability to communi-

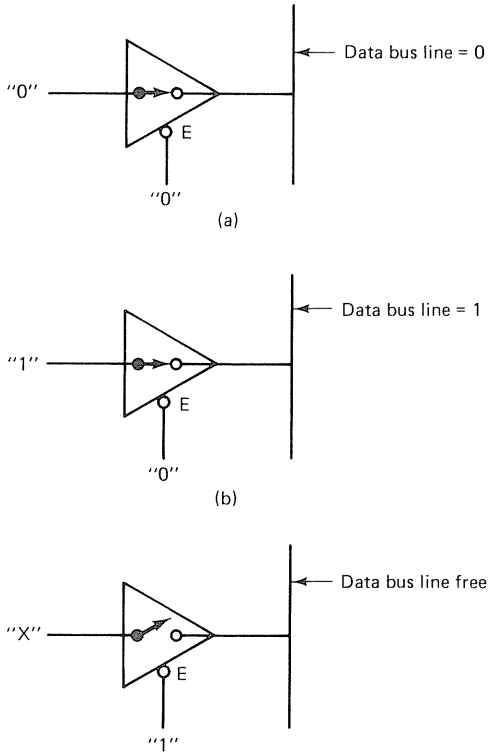



Figure 4-1 Tri-state gates are commonly used to gate data onto a data bus. In (a) and (b) the tri-state gate is *enabled* and the data bus line is connected to the input logic level. In (c) the tri-state gate is *disabled* and its output appears to be an *open circuit*. The data bus line is now free to be controlled by another driver on the line.

cate reliably with memory and I/O devices. In this application the tri-state gate is often called a *buffer*.

### Typical Input Port

Figure 4-2 illustrates the basic hardware requirements for a microcomputer input port. In this example, when the low-order address bus contains a $31_{10}$ (00011111) AND $\overline{IN}$ is active, the 7432 OR gate output will go low, enabling the tri-state gates. The microcomputer will now read the data established by switches S1 through S8. Note that *only one* BASIC command will cause this sequence of events to occur: Y=INP(31).

Although the circuit in Fig. 4-2 uses the 74LS244 octal buffer, other tri-state circuits can be used for input ports. Figure 4-3 lists several common circuits. Note that the 74LS241 and 74LS244 are particularly useful with microcomputers because they contain *eight* gates in one 20-pin package.

### The Data Input Is Always 8 Bits

As was true with the output port, the microcomputer always works with 8 bits of data. In the case of an input port, this means that 8 bits of data will

| IC | +5 V | GND |
|---|---|---|
| 7404 | 14 | 7 |
| 7430 | 14 | 7 |
| 7432 | 14 | 7 |
| 74LS244 | 20 | 10 |

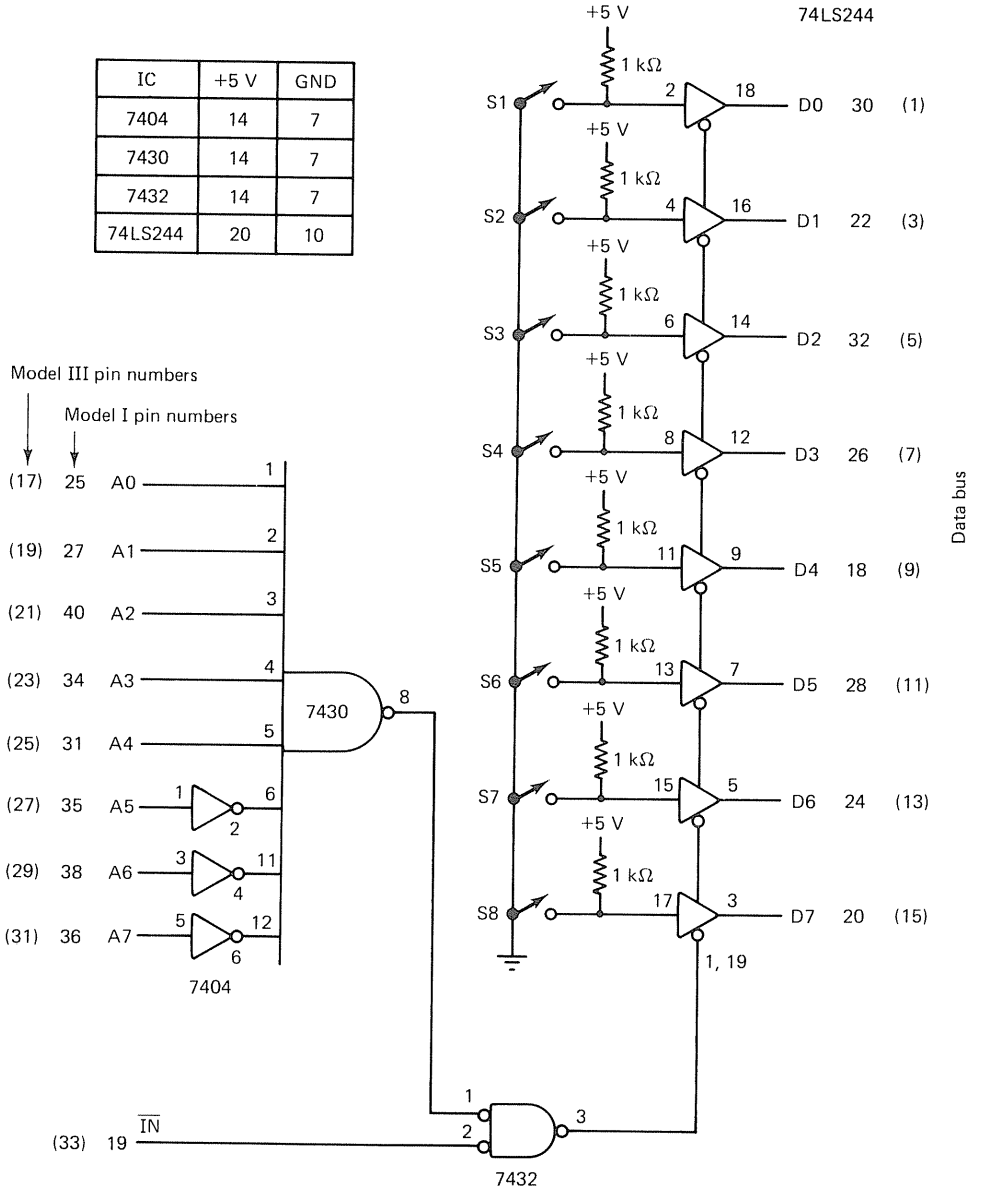



Figure 4-2 Basic microcomputer input port. The port address is $31_{10}$ (00011111). When the tri-states are enabled, the data byte established by S1 through S8 is gated onto the data bus.

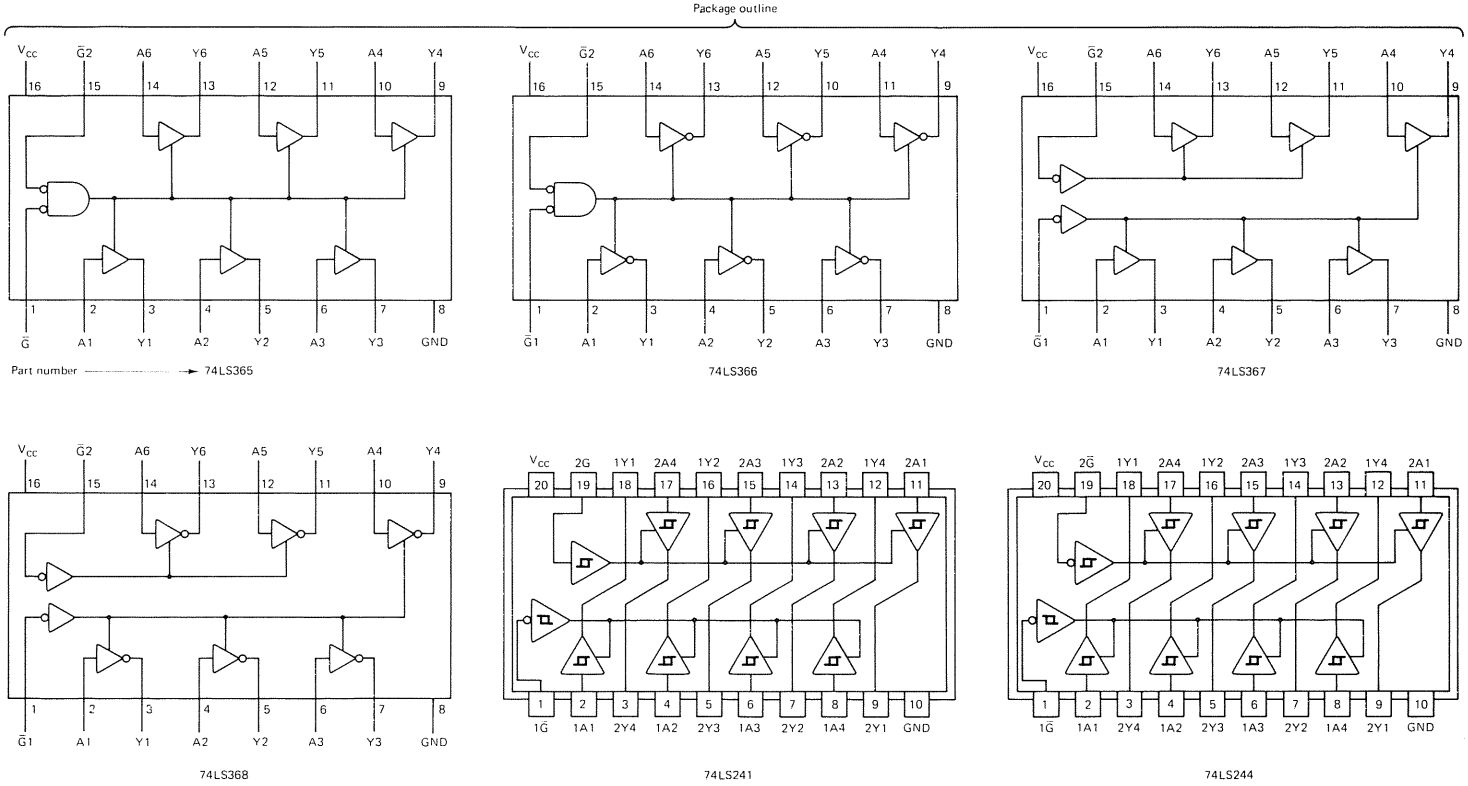



Figure 4-3 Typical tri-state buffers suitable as microcomputer input ports.

always be read even if the input port itself supplies *fewer* than eight lines. What will the computer "see" for those unused input lines? That is a good question and there is no specific answer. Generally, a line with no connection to it will appear to be a logic 1. This means that if the command **PRINT INP(0)** is given, the result will *probably* be 255 (assuming that there is no port 0 and $11111111_2 = 255_{10}$). However, microprocessors can be very unpredictable when connected to open circuits and may not always interpret an open line as a 1. For this reason it is best to *mask off* any bits whose value you do not want to consider in your program.

## Masking

The concept of masking is similar to designing a digital "filter" for logic 1's and 0's such that only the particular bits we wish to examine can penetrate the filter.

You might want to try the following program on your TRS-80.

```
 5 CLS
10 PRINT "ENTER TWO INTEGERS BETWEEN 0 AND 255"
20 INPUT "A= ";A
30 INPUT "B= ";B
40 PRINT "A AND B= ";A AND B
50 GOTO 10
```

Can you figure out why the program outputs what it does? For example, if A = 47 and B = 8, the response is **A AND B=8**. But if A = 47 and B = 16, the response is **A AND B=0**!

It may be helpful for you to refer to your TRS-80 *BASIC Language Reference Manual* section on logical operators. When the BASIC command **A AND B** is executed, a *bit-by-bit* comparison of the two operands is performed. To explain the puzzle in the preceding paragraph, let us convert 47, 8, and 16 to binary. For the first case,

$$\begin{array}{rcrl} 47 & = & 101111 & \text{(A)} \\ \cdot\ \underline{8} & = & \underline{\cdot\ 001000} & \text{(B)} \\ 8 & = & 001000 & \end{array}$$

and for the second case,

$$\begin{array}{rcrl} 47 & = & 101111 & \text{(A)} \\ \cdot\ \underline{16} & = & \underline{\cdot\ 010000} & \text{(B)} \\ 0 & = & 000000 & \end{array}$$

Note that the • signifies the AND operation. When the bits are ANDed together, a 1 results if and only if *both* bits are a 1. Also note that we can force 0's to appear in the result wherever they appear in word B.

This last result is the key to masking. Suppose that the 8 bits of data we input from a port correspond to various sensors within your home. This is illustrated in Fig. 4-4. Assume further that we would like to write a BASIC program that will determine if the furnace is *ON* or *OFF*. Because the furnace sensor is connected to bit D1, we need only monitor this bit and ignore or *mask* the other 7 bits.

Figure 4-5 illustrates the technique. The undesired bits are shown as "don't cares" or X's. The mask is chosen to force all these bits to become 0 except for D1. After performing the AND operation, it is a simple matter to determine the status of bit D1. For example, if the input port is 3:

```
 5 CLS
10 Y=INP(3)
20 IF (Y AND 2)=2 THEN A$="ON " ELSE A$="OFF"
30 PRINT@465, "THE FURNACE IS ";A$
40 GOTO 10
```

As another example, suppose that we wish to know if the front door, kitchen window, or bedroom window sensors are active. Referring to Fig. 4-4,

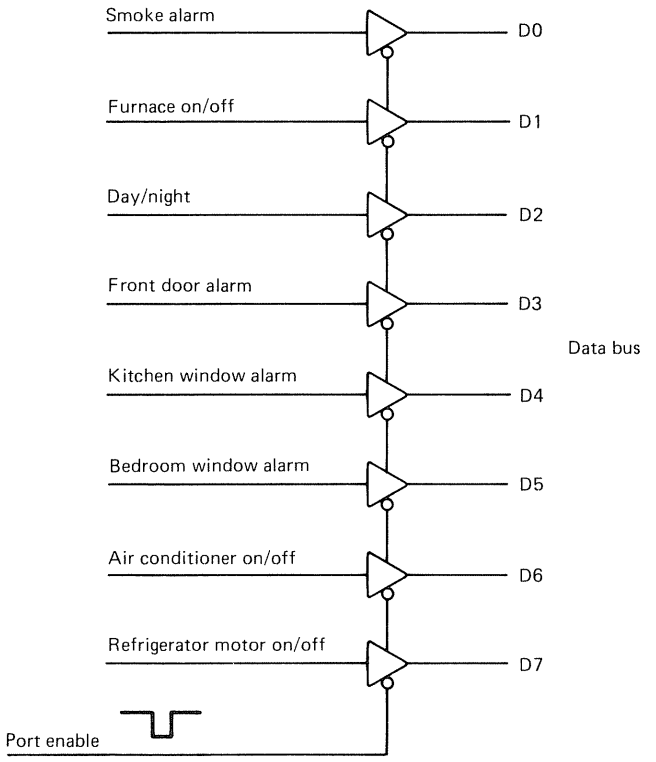



Figure 4-4 The 8 bits of a typical input port may represent conditions within your home. The electronics associated with each sensor is not shown (see Experiment 8).

| | | | | | |
|---|---|---|---|---|---|
| Input value | XXXX | X | X | D1 | X |
| Mask | 0000 | 0 | 0 | 1 | 0 |
| Result | 0000 | 0 | 0 | D1 | 0 |

If the result = 2 then the furnace is ON
If the result = 0 then the furnace is OFF

Figure 4-5 Masking to determine the status of bit D1. When ANDed with the input value, the mask forces all bits except D1 to be 0. The result is either 0 or 2.

this translates to D3 or D4 or D5 being high. In this case lines 20 and 30 are changed to

```
20 IF (Y AND 8)=8 OR (Y AND 16)=16 OR (Y AND 32)=32
   THEN A$="ALARM   " ELSE A$="NO ALARM"
30 PRINT@465, A$
```

where the masks for bits D3, D4, and D5 are 8, 16, and 32, respectively.

## PROCEDURE

*Note to Model III users.* The tri-state gates shown in Fig. 4-2 are *not needed* with your computer because the Model III data bus is brought out through a bi-directional buffer. The direction of data through this buffer is controlled by pin 43, $\overline{\text{EXT I/O SEL}}$. This line should be high to output data and low to input data. Tri-states are required if *more than one* port is connected to the expansion bus, however.

**Step 1.** Refer to Fig. 4-2 and carefully wire this circuit on your breadboard. Position the DIP switch upside down with switch 1 on your right. In this way a switch in the up position will correspond to a logic 1.

*Note to Model III users. Do not* use the 74LS244. Connect the port enable signal (7432 pin 3) to $\overline{\text{EXT I/O SEL}}$ pin 43 of the socket connector. Connect the switch outputs directly to D0 through D7.

**Step 2.** Test your circuit by loading and running the following program.

```
10 CLS
15 OUT 236,16        :REM MODEL III ONLY
20 Y=INP(31)
30 PRINT Y
40 GOTO 20
```

**Question 4-1.** Explain the result obtained in step 2.

**Question 4-2.** Modify the program so that it prints the following message: THE CURRENT VALUE OF THE SWITCHES IS: XXX. Make your modification such that the message is printed only *once* for any switch setting.

**Question 4-3.** What mask is needed to test bit D5?

**Step 3.** Load the following program into your computer. Try both positions for all eight switches for a given bit test. Experiment with the bit tested to be sure that you understand this concept.

```
10 CLS
20 INPUT "WHICH BIT DO YOU WISH TO TEST (0-7) ";B
30 M=2↑B                                    :REM M IS THE MASK
35 OUT 236,16                               :REM MODEL III ONLY
40 IF (INP(31) AND M)=M THEN
   A$="ON" ELSE A$="OFF"
50 PRINT@470, "SWITCH ";B;" IS ";A$
60 PRINT@534, "THE MASK VALUE IS ";M
70 GOTO 35
```

**Question 4-4.** Write a program to detect when bits 6 AND 3 = 1 AND when bits 0 OR 4 OR 7 = 1. When the condition is true, the program should display "OK"; if not true, the screen should be blank.

**Question 4-5.** Write a program to provide a *status report* of all 8 bits of input port 31 (simulated by the eight-position DIP switch). This program should continually update the display, providing an instantaneous status of all 8 bits.

**Step 4.** Reread the objectives listed at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

## SOLUTIONS TO QUESTIONS

4-1. You should see the decimal value of the switches rapidly scrolling by on the screen. As the switch settings change, the value on the screen will also change.

4-2. One possible way is:

```
10 CLS
20 OUT 236,16                               :REM MODEL III ONLY
25 Y=INP(31)
30 PRINT "THE CURRENT VALUE OF THE
   SWITCH IS ";Y
40 IF INP(31)=Y THEN 40
50 GOTO 10
```

4-3. 32.

4-4. One possible solution is:

```
10 CLS
15 OUT 236,16                                  :REM MODEL III ONLY
20 Y=INP(31)
30 IF (Y AND 64)=64 AND (Y AND 8)=8 AND
   ((Y AND 1)=1 OR (Y AND 16)=16 OR
   (Y AND 128)=128) THEN PRINT@480,
   "OK" ELSE 10
40 GOTO 15
```

4-5. One possible solution is:

```
 10 CLS
 20 FOR J=0 TO 7
 30 PRINT@(15+J*128), "SWITCH ":J;" IS: "
 40 NEXT J
 45 OUT 236,16                                 :REM MODEL III ONLY
 50 Y=INP(31)
 60 FOR J=0 TO 7
 70 IF (Y AND(2↑J))=(2↑J) THEN A$="ON "
    ELSE A$="OFF "
 80 PRINT@(31+J*128), A$
 90 NEXT J
100 GOTO 45
```

# EXPERIMENT 5

## I/O-MAPPED I/O USING THE 8255

### OVERVIEW

In this experiment you will wire an 8255 programmable peripheral interface (PPI) to the TRS-80 I/O bus. You will learn the software techniques needed to initialize the PPI, and a *traffic light controller* utilizing this circuit will be designed and demonstrated.

### OBJECTIVES

The key points to be learned from this experiment are:

1. Peripheral interface chips such as the 8255 have intelligence of their own but must be *programmed* by the microcomputer.
2. The 8255 programmable peripheral interface meets all the requirements for input and output ports, and no external latches or tri-state gates are required.
3. The 8255 contains three separate ports which may be programmed as input or output ports. A fourth port, called the *control port*, is used to control the mode of operation of the other three data ports.

### PARTS LIST

1 8255 programmable peripheral interface (Jameco INS 8255)
1 7430 8-input NAND gate
2 7404 hex inverters
1 7476 dual JK flip-flop

2 red LEDs
2 green LEDs
2 yellow LEDs
6 180-Ω resistors (brown-gray-brown)
2 1-kΩ resistors (brown-black-red)
2 pushbutton normally open switches

## DISCUSSION

### A "Smart" I/O Device

The 8255 is a *programmable peripheral interface* (PPI) integrated circuit. It has three I/O ports (or 24 I/O pins) that may be programmed by the microcomputer to be any combination of input and output ports. In fact, one of the ports, port C, can be split in half with four of the pins programmed as inputs, and the remaining four pins as outputs.

A block diagram of this chip is shown in Fig. 5-1. The three I/O ports can readily be seen (labeled ports A, B, and C) as well as two control blocks, called *group A control* and *group B control*. A unique feature of this circuit is that the microcomputer can communicate with these two control blocks and tell the PPI what I/O configuration is desired. For example, we might specify port A to be an input port (tri-state gates), port B to be an output port (latches), the high-order bits of port C to be outputs, and the low-order bits of port C to be inputs. And we can do all this by simply sending the proper *control word* to the two control blocks within the 8255. If a new port configuration is desired, we need only change the control word. No *rewiring* is required. And there is no need to provide external latches or tri-state gates, as these are all provided *internally*.

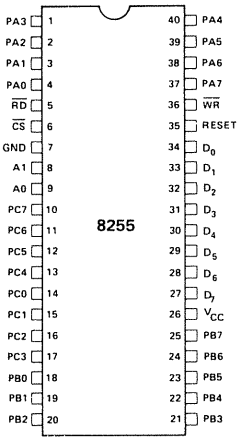
As can be seen in Fig. 5-1, the 8255 is a 40-pin device and requires a single +5-V power source.

### Modes of Operation

The 8255 may be programmed to operate in one of three modes, referred to as *mode 0*, *mode 1*, and *mode 2*. Mode 0 provides for basic input and output operations with the three ports A, B, and C. This is referred to as *unconditional* I/O in that the microcomputer outputs or inputs data without regard for the I/O port's *BUSY/READY* status. So far in this book none of the circuits discussed has required this status line. The I/O port is always assumed "ready" to receive or input data.

In some cases, however, the I/O device may have a special status line to indicate that it is busy right now and *cannot* accept or provide data. In this

PIN CONFIGURATION




PIN NAMES

| | |
|---|---|
| $D_7$–$D_0$ | DATA BUS (BI-DIRECTIONAL) |
| RESET | RESET INPUT |
| $\overline{CS}$ | CHIP SELECT |
| $\overline{RD}$ | READ INPUT |
| $\overline{WR}$ | WRITE INPUT |
| A0, A1 | PORT ADDRESS |
| PA7-PA0 | PORT A (BIT) |
| PB7-PB0 | PORT B (BIT) |
| PC7-PC0 | PORT C (BIT) |
| $V_{CC}$ | +5 VOLTS |
| GND | Ø VOLTS |

8255 BLOCK DIAGRAM

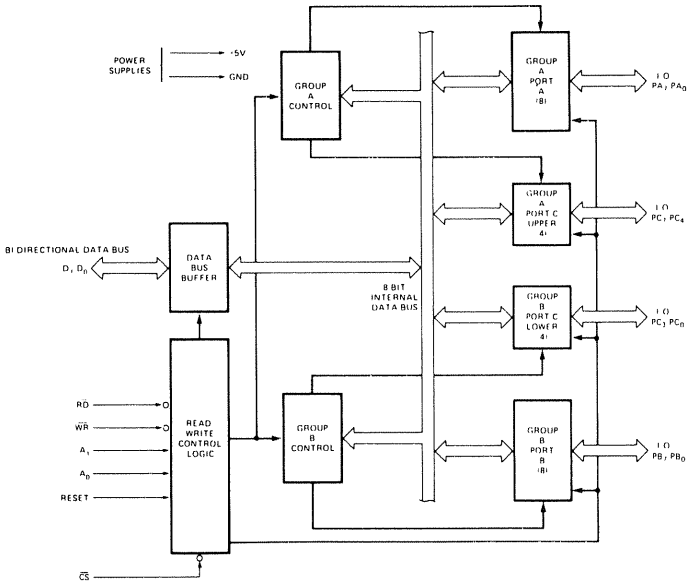



Figure 5-1 Block diagram of the 8255. This chip contains three separate I/O ports which may be programmed as inputs or outputs by the microcomputer. (Courtesy of Intel Corporation.)

case, the microcomputer must monitor this *BUSY/READY* status flag and attempt to input or output data from the selected I/O device only when it is ready. This particular mode of operation is also accommodated by the 8255 and is referred to as mode 1 or *strobed* input/output. In this mode ports A and B are used for the data paths, but port C is used to generate or accept the *BUSY/READY* or *handshaking* status signals. We examine this type of I/O in detail in Experiment 12.

The last mode of operation, mode 2, is similar to mode 1. The difference is that the data path consists of one set of I/O lines (port A) which are now configured to be *bidirectional*. Port C again provides handshaking capabilities. This mode of operation is intended for I/O devices that communicate over a single 8-bit bidirectional data bus.

In addition to the three modes of I/O discussed, there is also a *bit set/reset* mode. This mode allows individual bits of port C to be set or reset upon command from the microcomputer.

In this experiment we use only mode 0. Those readers interested in more detail about modes 1 and 2 are referred to the various data sheets supplied by Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051. In addition, more detail can also be found in *Microcomputer Interfacing with the 8255 PPI Chip* by Paul Goldsbrough (Indianapolis, Ind.: Howard W. Sams & Co., Inc., 1979).

## Interfacing to the 8255

When the 8255 is interfaced to the TRS-80, *all three* buses must be used. The eight data bus lines are connected directly to the 8255, as shown in Fig. 5-2. For I/O-mapped I/O, the $\overline{RD}$ and $\overline{WR}$ pins are connected to the TRS-80's $\overline{IN}$ and $\overline{OUT}$ control bus lines. Finally, the address bus has the two low-order address lines connected to the A0 and A1 inputs while the *chip select* input comes from an address decoder.

Because there are four different combinations of A0 and A1 (00, 01, 10, and 11), the 8255 actually appears to be *four* different I/O ports to the microcomputer. Of course, this is logical since there is a port A, B, C and a control port.

The specific port address is determined by the address decoder connected to the chip select ($\overline{CS}$) input. When this input goes low, the 8255 looks at A0 and A1 to determine which port you wish to talk with, and at its $\overline{RD}$ and $\overline{WR}$ inputs to see which *way* data should flow. In Fig. 5-2, the only address combinations that will cause the $\overline{CS}$ input to be low are 124 through $127_{10}$ (01111100 through 01111111).

**Example 5-1**

List all *commands* in BASIC that will access the PPI in Fig. 5-2.

**Solution** Because the 8255 uses the $\overline{IN}$ and $\overline{OUT}$ control lines, there are only two different commands: Y=INP(X) and OUT X,Y. For the specific addresses involved:

| COMMAND | ACTION |
|---|---|
| Y=INP(124) | input data from port A |
| Y=INP(125) | input data from port B |
| Y=INP(126) | input data from port C |
| Y=INP(127) | not allowed by the PPI |

and

| | |
|---|---|
| OUT 124,Y | output data Y to port A |
| OUT 125,Y | output data Y to port B |
| OUT 126,Y | output data Y to port C |
| OUT 127,Y | output data Y to the control port |

Note that the first four commands cause the $\overline{RD}$ input to be active (low) and the last four cause the $\overline{WR}$ input to be active.

| IC | +5 V | GND |
|---|---|---|
| 8255 | 26 | 7 |
| 7430 | 14 | 7 |
| 7404 | 14 | 7 |

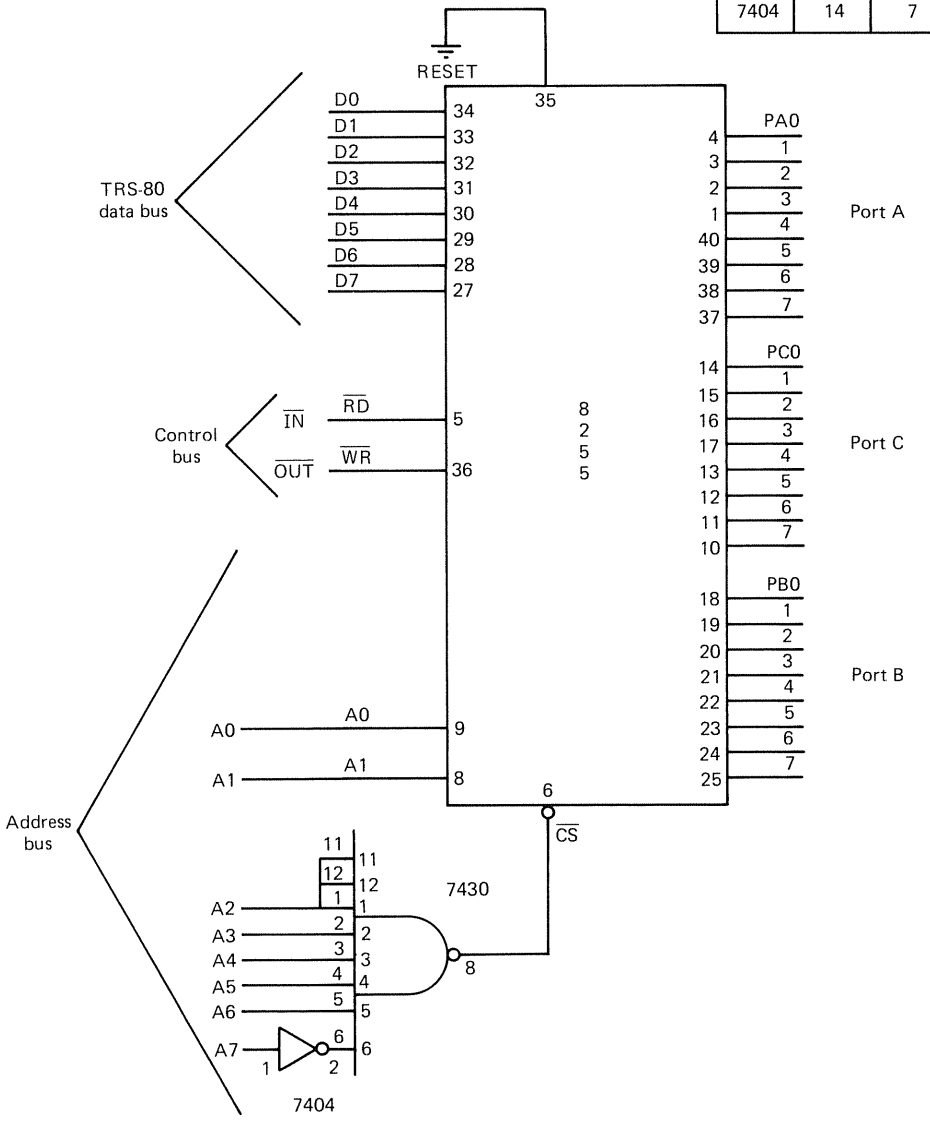



Figure 5-2 Typical interface circuit between the 8255 and TRS-80 I/O bus. In this circuit the 8255 occupies ports 124 through $127_{10}$. With a Model III computer, pin 43 on the I/O bus ($\overline{\text{EXT I/O SEL}}$) should also be connected to the 8255 pin 5 ($\overline{\text{RD}}$).

| A1 | A0 | Operation |
|---|---|---|
| 0 | 0 | Input data from port A |
| 0 | 1 | Input data from port B |
| 1 | 0 | Input data from port C |
| 1 | 1 | Illegal condition |

(a) $\overline{CS} = 0, \overline{RD} = 0$

| A1 | A0 | Operation |
|---|---|---|
| 0 | 0 | Output data to port A |
| 0 | 1 | Output data to port B |
| 1 | 0 | Output data to port C |
| 1 | 1 | Output data to control group |

(b) $\overline{CS} = 0, \overline{WR} = 0$

Figure 5-3 The 8255 truth table. In (a) the read (or input) operations are summarized; in (b) the write (or output) operations are indicated.

The commands listed in Example 5-1 actually represent a *truth table* for the 8255. This is repeated in Fig. 5-3 in a slightly more general format. One caution: Do not be misled by this table. Each of the I/O ports (A, B, and C) can be configured as an input or an output port, *but not both simultaneously*. The specific word written to the control group will determine this. This means that if port A is configured as an output port, we cannot input from this port even though the command **Y=INP(124)** can be given. We must rewrite the control word to cause port A to be an input port.

### Initializing the PPI from BASIC

Before using the PPI, it must be *initialized* for the specific I/O configuration desired. This must be done before any attempt is made to use the circuit in a system.

Initializing the PPI from BASIC is a simple matter consisting of a command of the form **OUT X,Y**, where X is the address of the PPI *control port* and Y is the desired *control word*. This word can be determined by referring to Fig. 5-4. This chart shows how each bit in the control word is established.

**Example 5-2**

Determine the control word needed for the following I/O configuration. Port A = output, port B = input, port C (PC0 through PC3) = input, and port C (PC4 through PC7) = output.

**Solution** Refer to Fig. 5-4:

| | |
|---|---|
| bit 7 = 1 | defines *mode* set versus bit set mode |
| bits 6, 5 = 00 | defines mode 0, unconditional I/O |
| bit 4 = 0 | port A = output |

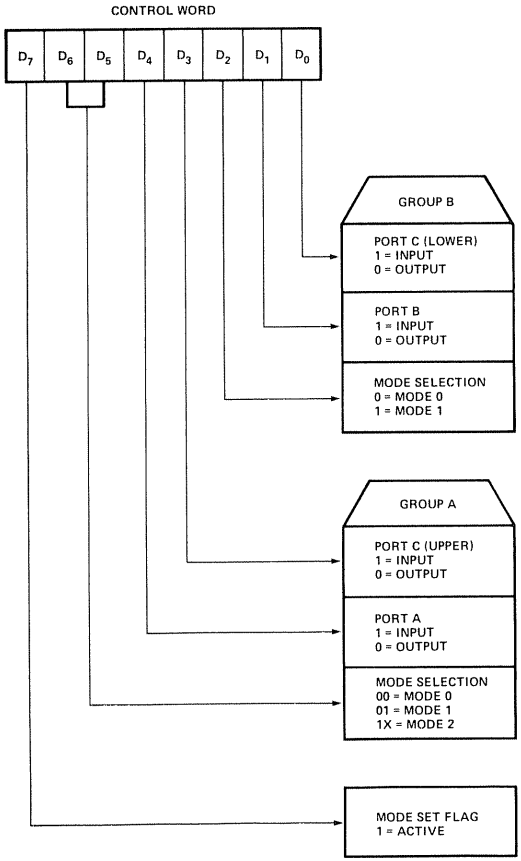



Figure 5-4 The 8255 mode definition format. The particular control word needed is determined by matching port definitions against those in the chart. Note that for modes 0 to 2 to be active, bit 7 must be a 1. (Courtesy of Intel Corporation.)

| | |
|---|---|
| bit 3 = 0 | port C (PC4 through PC7) = output |
| bit 2 = 0 | mode 0 |
| bit 1 = 1 | port B = input |
| bit 0 = 1 | port C (PC0 through PC3) = input |

The actual control word is 10000011 = $131_{10}$.

Using the hardware shown in Fig. 5-2 and the I/O requirements in Example 5-2, the PPI is initialized with the command

OUT 127,131

## PROCEDURE

**Step 1.** Refer to Fig. 5-5 and carefully wire this circuit on your breadboard. Do not wire the LEDs or flip-flops at this time. Use caution when

| IC | +5 V | GND |
|---|---|---|
| 8255 | 26 | 7, 35 |
| 7404(2) | 14 | 7 |
| 7476 | 5 | 13 |
| 7430 | 14 | 7 |

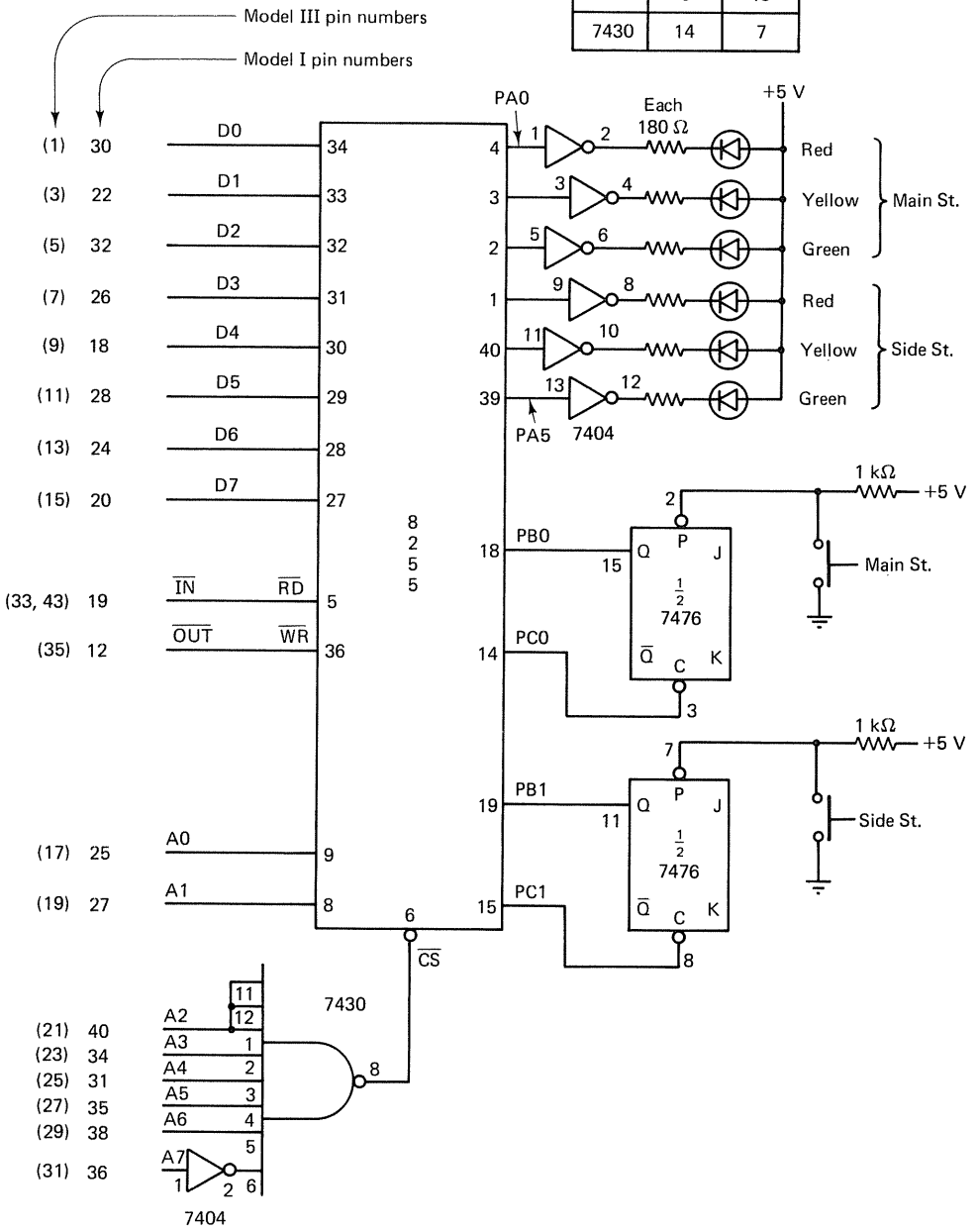



Figure 5-5 Hardware for the traffic light controller. Ports A and C are outputs while port B is an input. Cars are detected with two normally open pushbutton switches.

handling the 8255, as it is an MOS device and subject to damage due to *static electricity*. Do not handle the IC leads and ground yourself to your breadboard before placing the chip into the socket.

*Note to Model III users.* Wire this circuit as shown in Fig. 5-5, but also connect pin 43 ($\overline{\text{EXT I/O SEL}}$) on your socket connector to the 8255 pin 5 ($\overline{\text{IN}}$ should already be connected to this pin). In this way, whenever the Model III does an input, the special buffers in your computer will be configured in the correct direction.

**Step 2.** We will test the hardware by wiring ports A and B in parallel and then outputting data to port A and inputting the data from port B. Wire PA0 to PB0, PA1 to PB1, and so on. Refer to Fig. 5-2 for pin numbers.

**Question 5-1.** Write a BASIC program that will test this hardware arrangement. What is the control port address? What is the control word?

**Step 3.** To gain more experience with the 8255, repeat step 2 with ports B and C wired in parallel and using B as an input port and C as an output port.

*Note.* In the next steps we assemble and test the hardware needed to convert the TRS-80 into a *traffic light controller*. Green, red, and yellow LEDs will simulate the traffic lights and two pushbutton switches will act as car sensors.

**Step 4.** Remove all wires from ports A, B, and C. Connect the six LEDs and six 180-Ω resistors to port A using a *second* 7404 as shown in Fig. 5-5.

**Step 5.** Test your LED interface by loading and running the following program:

```
 5 OUT 236,16                    :REM MODEL III ONLY
10 OUT 127,130                   :REM MAKE PORT A AN OUTPUT
20 FOR J=0 TO 5
30 OUT 124,2↑J
40 FOR I=1 TO 250:   NEXT I
50 NEXT J
60 GOTO 10
```

**Question 5-2.** What output code word (decimal) is needed to show (a) red on Main St. and green on Side St., and (b) yellow on Main St. and red on Side St.?

**Step 6.** If your LEDs are working properly, add the car sensors consisting of one 7476 and two pushbutton switches. Refer to Fig. 5-5 for details.

**Question 5-3.** What control word is needed to initialize the 8255 for the circuit in Fig. 5-5?

*Note.* The 7476 JK flip-flop will be *set* by its asynchronous preset input each time a switch is depressed (simulating a car). Because the flip-flop can only set once, the computer must *reset* this flip-flop so that it can detect the *next* closure of the switch. We can do this in software by pulsing the PC0 and PC1 line from high to low and back to high again.

**Step 7.** Test your flip-flop circuit by loading and running the following program. See if you can *predict* what this program will do.

```
 10 CLS
 15 OUT 236,16                                    :REM MODEL III ONLY
 20 OUT 127,130                                   :REM INIT PPI
 30 OUT 126,255:  OUT 126,0:  OUT 126,255:        :REM RESET
    THE FLIP-FLOPS
 40 PRINT "PUSH THE MAIN ST PUSHBUTTON"
 50 IF (INP(125) AND 1)=1 THEN OUT 124,7 ELSE 50
 60 PRINT "PUSH THE SIDE ST PUSHBUTTON"
 70 IF (INP(125) AND 2)=2 THEN OUT 124,56 ELSE 70
 80 INPUT "PUSH ENTER TO RESET THE
    FLIP-FLOPS";A$
 90 OUT 124,0
100 GOTO 10
```

**Step 8.** When your hardware works successfully through step 7, you are ready to write the traffic light controller program. The following requirements should be met:

1. Main St. is the *priority* street. It must have a minimum of 30 s of green before changing to red. In addition, it should only change to red if 30 s have elapsed *and* at least one car is waiting on Side St.
2. The green light on Side St. should stay green until either 30 s has elapsed *or* three or more cars are waiting on Main St.
3. In all cases, lights should sequence from green, to yellow (for 4 s), to red.

*Note.* A solution to this step is provided at the end of this experiment.

**Step 9.** Reread the objectives as stated at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

*Note.* Do not disassemble this circuit, as portions of it will be used in Experiment 6.

5-1. The control port is 127. Because port C is undefined, there are four possible control words that configure A and B as desired. They are: 130, 131, 138, and 139. One possible test program is:

```
10 CLS
15 OUT 236,16                                  :REM MODEL III ONLY
20 OUT 127,130                                 :REM INIT PPI
30 INPUT "DATA TO OUTPUT TO PORT A IS ";A
40 OUT 124,A                                   :REM 124 IS PORT A
50 INPUT "PUSH ENTER TO SEE THE DATA
   INPUT FROM PORT B ";B$
60 PRINT INP(125)                              :REM 125 IS PORT B
70 IF INP(125)=A THEN PRINT "PORT TEST
   IS OK" ELSE PRINT "PORT TEST FAILS"
80 INPUT "AGAIN ";B$
90 IF B$="YES" THEN 30 ELSE END
```

Another test program that lets the TRS-80 do all the testing is:

```
1000 CLS
1005 OUT 236,16                          :REM MODEL III ONLY
1010 OUT 127,130                         :REM INIT PPI
1020 FOR J=0 TO 255
1030 OUT 124,J
1040 IF INP(125)=J THEN 1060
1050 PRINT "TEST FAILS ON OUTPUT ";J
1060 NEXT J
1070 PRINT "TEST OVER"
```

5-2. (a) 33; (b) 10.

5-3. Port A = output to LEDs. Port B = input from car sensors. Port C (lower) = output to reset car sensor. Control word = 1000X010 = $130_{10}$ or $138_{10}$.

STEP 8 (solution) A possible solution is shown below (also refer to the flowchart in Fig. 5-6).

```
10 OUT 236,16                      :REM MODEL III ONLY
20 OUT 127,130                     :REM INIT PPI
30 OUT 126,255:   OUT 126,0:       :REM RESET SENSORS
   OUT 126,255
40 OUT 124,12                      :REM MAIN IS GREEN SIDE IS RED
50 D=30:   GOSUB 500               :REM WAIT 30 S
60 IF (INP(125) AND 2)<>2          :REM CAR ON SIDE ST?
   THEN 60:
70 OUT 126,3:   OUT 126,1:         :REM RESET SIDE ST SENSOR
   OUT 126,3
```

```
 80  OUT 124,10                    :REM MAIN IS YELLOW SIDE IS RED
 90  D=4:   GOSUB 500              :REM WAIT 4 S
100  OUT 124,33                    :REM SIDE IS GREEN MAIN IS RED
110  C=0                           :REM CAR COUNTER
120  FOR J=1 TO 30                 :REM 30 LOOPS AT 1 S EACH
130  IF (INP(125) AND 1)=1         :REM CAR ON MAIN?
     THEN 160:
140  D=1:   GOSUB 500              :REM DO LOOPS AT 1 S EACH
150  GOTO 200
```

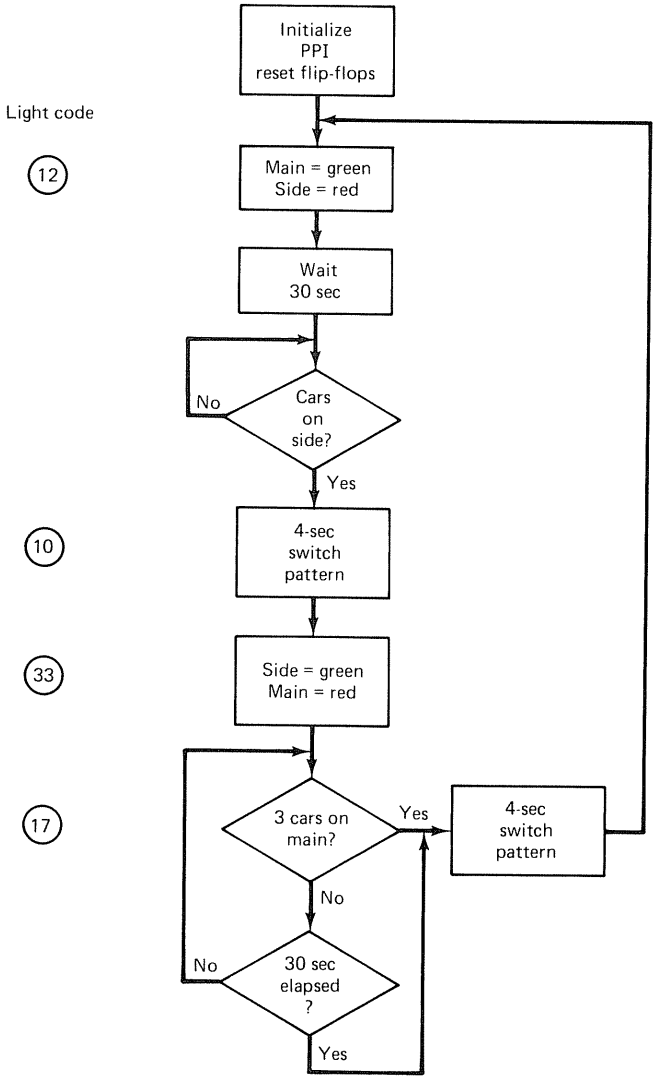



Figure 5-6 Traffic light controller flowchart.

```
160  OUT 126,3:   OUT 126,2:        :REM RESET MAIN ST SENSOR
     OUT 126,3
170  C=C+1                          :REM CAR COUNTER
180  IF C<3 THEN 200                :REM 3 CARS?
190  J=30
200  NEXT J
210  OUT 124,17                     :REM SIDE IS YELLOW MAIN IS RED
220  D=4:   GOSUB 500               :REM WAIT 4 S
230  GOTO 40
500  FOR T=1 TO D*360               :REM TIME DELAY, ADJUST FOR
                                     1 SEC PER VALUE OF D
510  NEXT T
520  RETURN
```

*Note.* Although this program should work with you hardware, erratic operation may occur if your switches have excessive *bounce time*. This is because even though the computer will try to reset the flip-flop, if the switch bounces too long, it will set again. The cure is a brief time delay in lines 65 and 160:

```
FOR I=1 TO 5:   NEXT I
```

This allows the switch to stop bouncing *before* trying to reset the flip-flop.

# EXPERIMENT 6

## MEMORY-MAPPED I/O USING THE 8255

*Note.* This experiment *cannot be done* on a Model III TRS-80. This is because the high-order address lines and $\overline{RD}$ and $\overline{WR}$ control lines are not brought out to the Model III expansion connector. The experiment also cannot be done using the expansion interface and a Model I computer. This is because the $\overline{RD}$ and $\overline{WR}$ control signals are not enabled on the expansion interface bus (you may want to disconnect the expansion interface temporarily for this experiment).

### OVERVIEW

In this experiment you will wire the 8255 programmable peripheral interface chip to the TRS-80 using a *memory-mapped* I/O technique. The traffic light controller developed in Experiment 5 will be modified to use the two memory-mapped I/O commands **PEEK** and **POKE**.

### OBJECTIVES

The key points to be learned from this experiment are:

1. When using memory-mapped I/O, I/O devices appear to be *memory locations* to the microcomputer.
2. The control bus signals $\overline{RD}$ and $\overline{WR}$ are used in place of the $\overline{IN}$ and $\overline{OUT}$ signals used for I/O-mapped I/O.
3. When decoding a memory-mapped I/O address, all *16* address lines must be used for complete decoding.
4. BASIC communicates with memory-mapped I/O devices through the **PEEK** and **POKE** commands.

PARTS LIST

| | |
|---|---|
| 1 | 8255 programmable peripheral interface (JAMECO INS P255) |
| 1 | 7430 8-input NAND gate |
| 1 | 7404 hex inverter |
| 1 | 7476 dual JK flip-flop |
| 2 | red LEDs |
| 2 | green LEDs |
| 2 | yellow LEDs |
| 6 | 180-Ω resistors (brown-gray-brown) |
| 2 | 1-kΩ resistors (brown-black-red) |
| 2 | normally open pushbutton switches |

DISCUSSION

Typical Memory-Mapped I/O Ports

Not all microcomputers have I/O-mapped I/O capabilities. In fact, most do not. Microprocessors such as the 6502 and 6800 support only *memory-mapped* I/O. The TRS-80 uses the Z-80 microprocessor, which, like its predecessor the 8080, has *both* memory-mapped and I/O-mapped I/O capabilities.

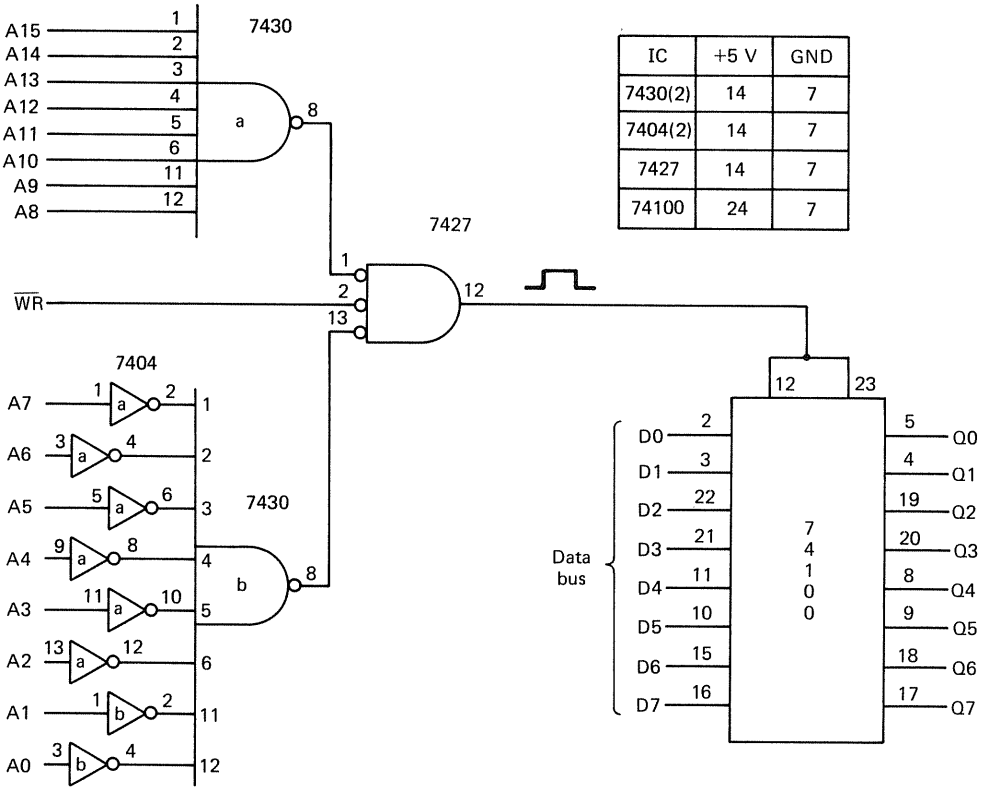
What is the difference between these two I/O techniques? Figure 6-1 illustrates a typical memory-mapped output port. One of the most obvious differences is that the decoder must decode the full *16*-bit address. This is what makes the port appear to be a memory location to the microcomputer. In fact, because it has a memory address, the microcomputer thinks it is a memory location! This is because it is interfaced to the computer in exactly the same way that a memory cell would be. Note that the memory write ($\overline{WR}$) control signal is used to enable the output port. Whenever the microcomputer does a memory write to address 65280 (1111111100000000), the data bus contents will be caught and latched by the 74100.

How do we instruct BASIC to do such an operation? We use the **POKE X,Y** command. For example, **POKE 127,3** will output a 3 (00000011) to memory location $127_{10}$.

Referring to Fig. 6-1, it would appear that we could send data to the 74100 latch with a BASIC command such as

**POKE 65280,3**

However, integers in TRS-80 BASIC must be in the range −32768 to +32767 (this allows signed arithmetic with integers). For this reason, addresses




| IC | +5 V | GND |
|---|---|---|
| 7430(2) | 14 | 7 |
| 7404(2) | 14 | 7 |
| 7427 | 14 | 7 |
| 74100 | 24 | 7 |

Figure 6-1 Memory-mapped output port. The port address is 65280 and the data is output with the command **POKE -256,Y.**

greater than 32767 must be converted with the following formula:

new address = desired address - 65536

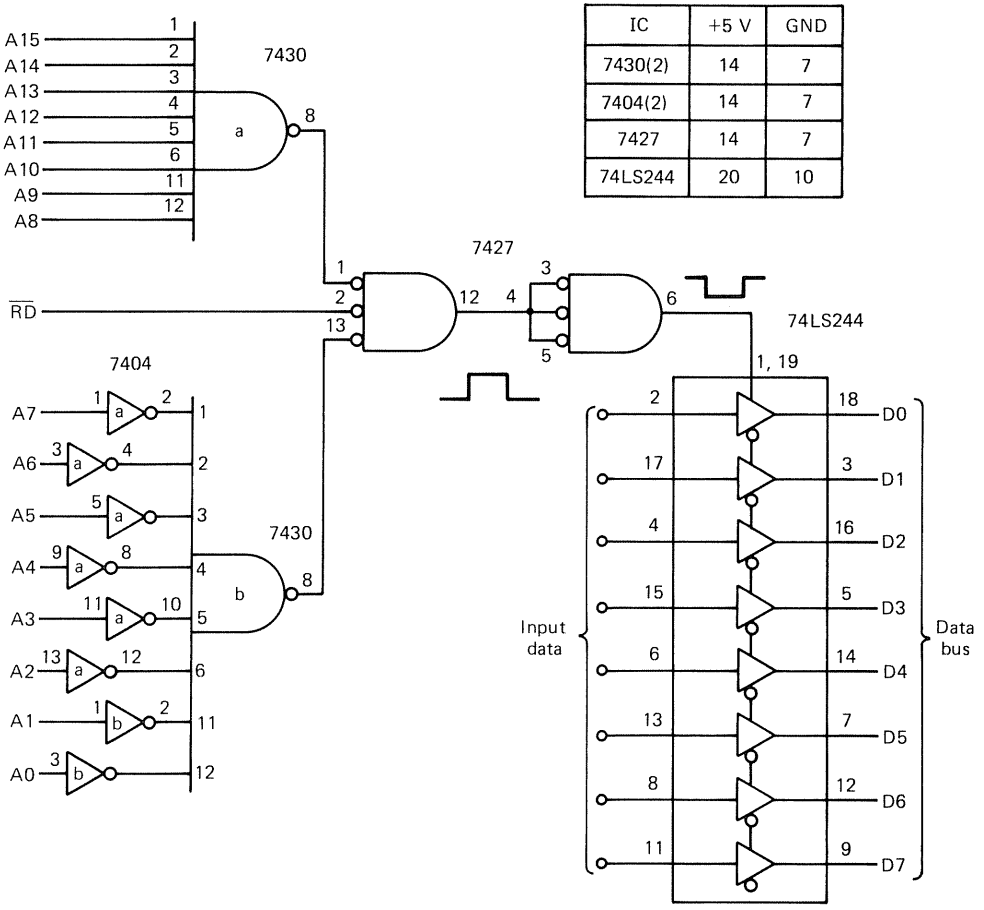
This means that to write data (3 in this example) to memory location 65280, the following command would be used:

**POKE -256,3**

where 65280 - 65536 = -256.

Memory-Mapped Input Port

Figure 6-2 illustrates the same decoder circuit from Fig. 6-1 now used to enable a memory-mapped *input port*. Note that the memory read ($\overline{RD}$) control signal is used in this case. From BASIC we may input data from a




| IC | +5 V | GND |
|---|---|---|
| 7430(2) | 14 | 7 |
| 7404(2) | 14 | 7 |
| 7427 | 14 | 7 |
| 74LS244 | 20 | 10 |

Figure 6-2 Memory-mapped input port. The port address is 65280 and the data is input with the command **Y=PEEK(-256)**.

memory-mapped input port with the command **Y=PEEK(X)**, where X must be calculated with the formula

new address = desired address - 65536

when X is greater than 32767.

Referring to Fig. 6-2, the command

**Y=PEEK(-256)**

will cause the $\overline{RD}$ line to go low and the address bus to contain 1111111100000000. This will enable the 74LS244 tri-state gates and pass the input data to the microcomputer.

Table 6-1 summarizes the four types of I/O operations and indicates the

TABLE 6-1 COMPARING THE FOUR TYPES OF I/O

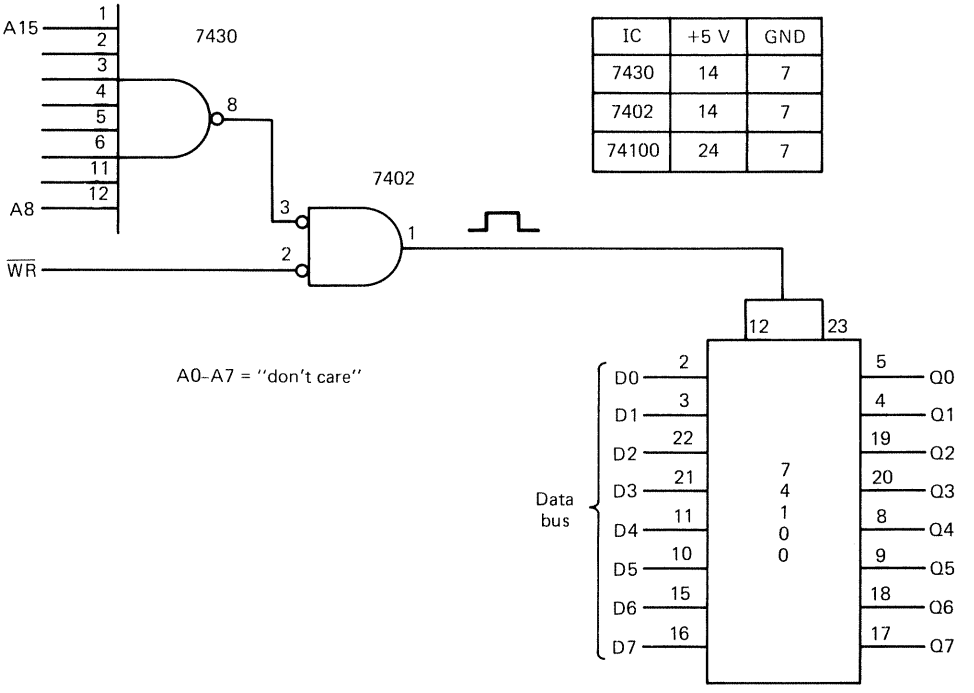
| | Input port | | Output port | |
|---|---|---|---|---|
| | Memory mapped | I/O mapped | Memory mapped | I/O mapped |
| Decoder | 16 bits | 8 bits (A0–A7) | 16 bits | 8 bits (A0–A7) |
| Control bus | $\overline{RD}$ | $\overline{IN}$ | $\overline{WR}$ | $\overline{OUT}$ |
| Basic command | Y=PEEK(X)[a] | Y=INP(X) | POKE X,Y[a] | OUT X,Y |

[a]For addresses greater than 32767, use address = desired address − 65536.

significant differences. Remember that in all cases, the data, Y, is an 8-bit quantity and is therefore limited to the range 0 through 255.

## Partial Decoding

Partial decoding is often used with memory-mapped I/O to simplify the decoding circuitry. For example, the memory-mapped output port in Fig. 6-3 using a partial address decoder. Because address lines




| IC | +5 V | GND |
|---|---|---|
| 7430 | 14 | 7 |
| 7402 | 14 | 7 |
| 74100 | 24 | 7 |

Figure 6-3 Partially decoded memory-mapped output port. This circuit will respond to memory writes in the range 65280 through 65535.

A0 through A7 are no longer tested, these lines become "*don't cares.*" There are now 256 possible memory locations that this circuit will respond to, from 65280 (1111111100000000) to 65535 (1111111111111111). The advantage is that the decoder has been simplified (one less gate is required) but the disadvantage is that one port now occupies the memory space of 256 locations.

### I/O Mapped Versus Memory Mapped—Which Is Best?

As indicated in Table 6-1, when using BASIC the *software* differences between memory-mapped and I/O-mapped I/O are minimal. If anything, memory-mapped I/O is a bit more awkward due to having to calculate the memory address for ports above 32767.

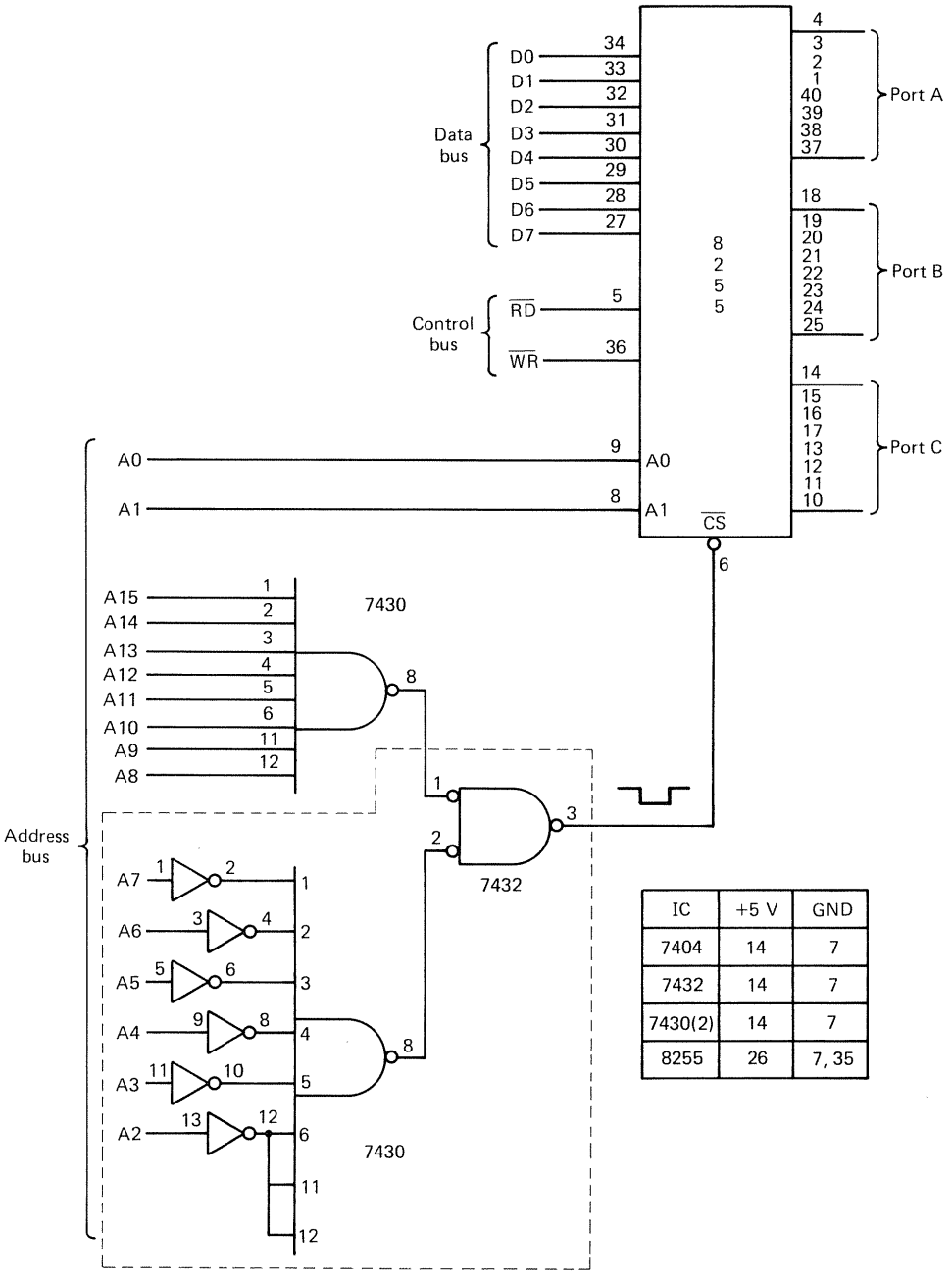
As far as *hardware* is concerned, memory-mapped I/O is definitely more complex, since an additional eight address lines (A8 through A15) must be decoded. In addition, care must be taken when choosing the memory-mapped I/O address. For example, a 16K TRS-80 (16K RAM) uses all memory space from 0 through 32767, and therefore no memory-mapped ports may use this memory space. In a 48K system, *all* the computer's memory space is occupied and memory-mapped I/O is not possible. (This is apparently the reason the Model III computer does not support memory-mapped I/O. It is internally expandable to 48K, in which case there would be no room for any user memory-mapped I/O ports.)

What, then, is the advantage of memory-mapped I/O? When viewed from BASIC, there really are no advantages. However, when programming in the microprocessor's *assembly language*, there are numerous instructions that may be used to access memory-mapped I/O devices compared to only a few that can be used for I/O-mapped I/O. This means that it is easier to handle memory-mapped I/O ports than I/O-mapped ports when working in assembly language.

### Using the 8255

Figure 6-4 illustrates a technique in which the 8255 may be interfaced to the TRS-80 I/O bus in a memory-mapped configuration. The principle difference between this circuit and the I/O-mapped circuit in Fig. 5-2 is that the chip select input ($\overline{CS}$) is enabled by a decoder of bits A2 through A15 representing the *full* address bus. In addition, the $\overline{RD}$ and $\overline{WR}$ control signals are used instead of $\overline{OUT}$ and $\overline{IN}$. As we have discussed previously, the 8255 will occupy four ports or addresses; with the circuit shown in Fig. 6-4, these are:

| | |
|---|---|
| 65280 | port A |
| 65281 | port B |
| 65282 | port C |
| 65283 | control port |




| IC | +5 V | GND |
|---|---|---|
| 7404 | 14 | 7 |
| 7432 | 14 | 7 |
| 7430(2) | 14 | 7 |
| 8255 | 26 | 7, 35 |

Figure 6-4 Memory-mapped 8255 interface. This circuit occupies memory locations 65280 through 65283. If the gates within the dashed lines are removed, a partial decoding scheme results.

This circuit could be simplified somewhat by removing the gates within the dashed lines in Fig. 6-4. This would result in a partial decoding scheme and the 8255 would now occupy 256 consecutive memory locations (64 *sets* of 4) from 65280 (1111111100000000) through 65535 (1111111111111111). This amounts to the last 256-byte page in the TRS-80 computer. Because memory additions are usually done in 1K, 4K, or 16K blocks, this partially decoded interface may be a good choice when a large block of memory is left open.

**Example 6-1**

Write a BASIC program to initialize the 8255 in Fig. 6-4 so that ports A, B, and PC0 through PC3 are inputs and port PC4 through PC7 is an output.

**Solution** The first step is to determine the control word. Referring to Fig. 5-4, this is

$$10010011 = 147_{10}$$

This word must be written to the control port at address 65283. Recall that the actual address must be calculated as

$$\text{address} = 65283 - 65536 = -253$$

The program then consists simply of the command

```
POKE(-253),147
```

## PROCEDURE

*Note.* The hardware required for this experiment is *identical* to that shown in Fig. 5-5. Two modifications will be required, however, First, the $\overline{IN}$ and $\overline{OUT}$ control lines are replaced by the $\overline{RD}$ and $\overline{WR}$ control lines, respectively. Second, the 7430 decoder will now decode the *high-order* address bus A8 through A15. This will result in a *partially* decoded circuit.

**Step 1.** Refer to Fig. 6-5 and wire this circuit on your breadboard. If you have done Experiment 5, this will require only the two changes listed in the note above.

**Question 6-1.** What makes this circuit *partially* decoded? What is the *lowest* address of the control port?

**Question 6-2.** What changes would be needed to make this circuit a *fully decoded* memory-mapped interface at address 65280 through 65283?

| IC | +5 V | GND |
|---|---|---|
| 8255 | 26 | 35, 7 |
| 7404 | 14 | 7 |
| 7476 | 5 | 13 |
| 7430 | 14 | 7 |

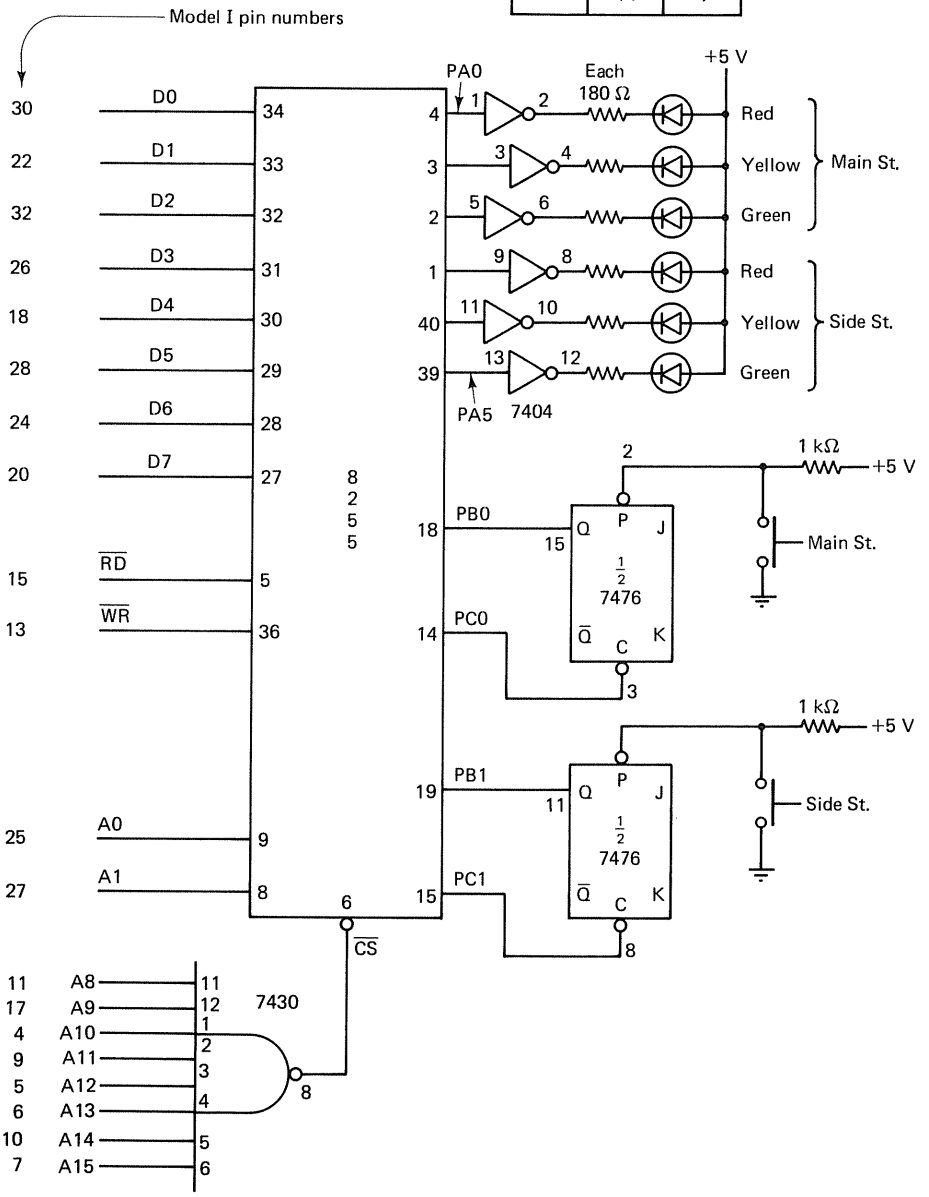



Figure 6-5 Memory-mapped version of the traffic light controller in Experiment 5. Compare this circuit to the one in Fig. 5-5.

**Step 2.** In Experiment 5, step 5, we developed a test program for the six LEDs. Try running the following memory-mapped version of this program.

```
10 POKE(-253),130                    :REM INIT PPI
20 FOR J=0 TO 5
30 POKE(-256),2↑J
40 FOR I=1 TO 250:  NEXT I
50 NEXT J
60 GOTO 10
```

**Question 6-3.** Test the flip-flop portion of the circuit by rewritting the test program in Experiment 5, step 7.

**Step 3.** Write a memory-mapped traffic light controller program similar to the one developed in Experiment 5. A solution is provided at the end of this experiment.

**Step 4.** Reread the objectives as stated at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

## SOLUTIONS TO QUESTIONS

**6-1.** Because address lines A2 through A7 are not decoded, this circuit is partially decoded. The control port address is 11111111XXXXXX11 = 65283, choosing the "don't cares" to be 0.

**6-2.** See Fig. 6-6.

**6-3.**

```
 10 CLS
 20 POKE(-253),130                          :REM INIT PPI
 30 POKE(-254),255:  POKE(-254),0:          :REM RESET THE
    POKE(-254),255                           FLIP-FLOPS
 40 PRINT "PUSH THE MAIN ST PUSHBUTTON"
 50 IF (PEEK(-255) AND 1)=1 THEN POKE(-256),7
    ELSE 50
 60 PRINT "PUSH THE SIDE ST PUSHBOTTON"
 70 IF (PEEK(-255) AND 2)=2 THEN POKE(-256),56
    ELSE 70
 80 INPUT "PUSH ENTER TO RESET THE
    FLIP-FLOPS";A$
 90 POKE(-256),0
100 GOTO 10
```

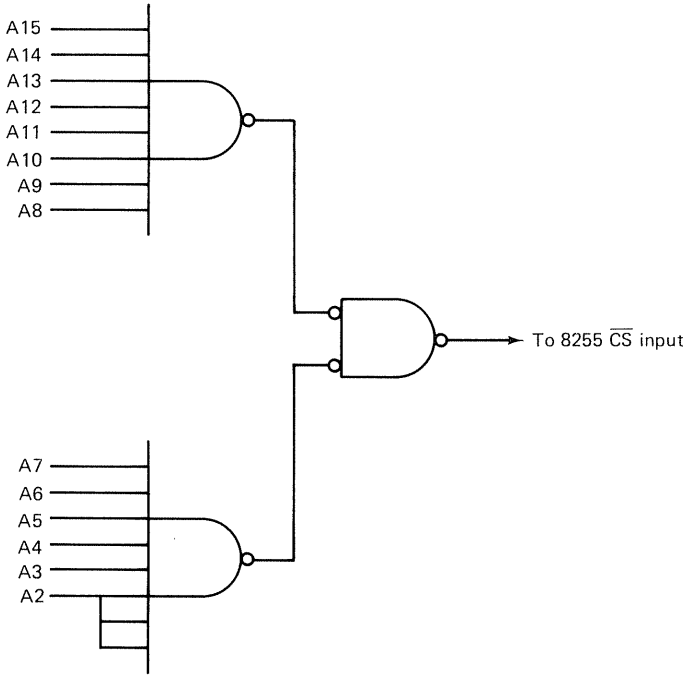



Figure 6-6 Solution to Question 6-2. All 16 address lines are now decoded.

STEP 3 (solution)

```
 20  POKE(-253),130                      :REM INIT PPI
 30  POKE(-254),255:  POKE(-254),0:      :REM RESET SENSORS
     POKE(-254),255
 40  POKE(-256),12                       :REM MAIN IS GREEN SIDE IS RED
 50  D=30:  GOSUB 500                    :REM WAIT 30 S
 60  IF (PEEK(-255) AND 2)<>2            :REM CAR ON SIDE ST?
     THEN 60
 70  POKE(-254),3:  POKE(-254),1:        :REM RESET SIDE ST SENSOR
     POKE(-254),3 RESET SIDE ST
     SENSOR
 80  POKE(-256),10                       :REM MAIN IS YELLOW SIDE IS RED
 90  D=4:  GOSUB 500                     :REM WAIT 4 S
100  POKE(-256),33                       :REM SIDE IS GREEN MAIN IS RED
110  C=0                                 :REM CAR COUNTER
120  FOR J=1 TO 30                       :REM 30 LOOPS AT 1 S EACH
130  IF (PEEK(-255) AND 1)=1             :REM CAR ON MAIN?
     THEN 160
```

```
140  D=1:  GOSUB 500                    :REM DO LOOPS AT 1 S EACH
150  GOTO 200
160  POKE(-254),3:  POKE(-254),2:       :REM RESET MAIN ST
     POKE(-254),3                        SENSOR
170  C=C+1                              :REM CAR COUNTER
180  IF C<3 THEN 200                    :REM 3 CARS?
190  J=30
200  NEXT J
210  POKE(-256),17                      :REM YELLOW ON SIDE MAIN IS
                                         RED
220  D=4:  GOSUB 500                    :REM WAIT 4 S
230  GOTO 40
500  FOR T=1 TO D*360                   :REM TIME DELAY, ADJUST FOR
                                         1 S PER VALUE OF D
510  NEXT T
520  RETURN
```

*Note.* Although this program should work with your hardware, erratic operation may occur if your switches have excessive *bounce* time. This is because even though the computer will try to reset the flip-flop, if the switch bounces too long it will set again. The cure is a brief time delay in lines 65 and 160:

```
FOR I=1 TO 5:  NEXT I
```

This allows the switch to stop bouncing *before* trying to reset the flip-flop.

# EXPERIMENT 7

## 1K STATIC MEMORY INTERFACE

*Note.* This experiment, for the same reasons as those given for Experiment 6, *cannot be done* on a Model III TRS-80 or a Model I computer with the expansion interface. You may still wish to read through it, however, to familiarize yourself with the concepts of static memory interfacing.

### OVERVIEW

In this experiment you will add an additional 1K of memory to your TRS-80. A memory test program using the **PEEK** and **POKE** commands is developed.

### OBJECTIVES

The key points to be learned from this experiment are:

1. Random access memory (*RAM*) may be read from or written to, as contrasted with read-only memory (*ROM*), which can only be read from. User programs are stored in *RAM*, while permanent operating system programs are stored in *ROM*.
2. The $\overline{RD}$ and $\overline{WR}$ control bus signals determine the *direction* of data flow to the memory.
3. A memory interface is identical to a memory-mapped I/O interface and requires an address decoder plus data bus and control bus connections.
4. The *2114* is a static, 1K $\times$ 4 *RAM* and two of these chips are required to add 1K of memory to the TRS-80.

PARTS LIST

2 2114 1K X 4 static RAMs (Radio Shack 276-2504)
1 7430 8-input NAND gate
1 7404 hex inverter
1 7400 quad NAND gate

DISCUSSION

### Types of Memory

Computer memory comes in two varieties: *ROM* and *RAM*. *Read-only memory* or *ROM* can be thought of as permanent and unchanging. When you turn on your TRS-80, it immediately begins running the BASIC interpreter that is stored in ROM. This is a very nice feature which you probably take for granted. However, computers without ROM require a "bootstrap" program to be loaded (via a front panel), which in turn allows BASIC or some other high-level language to be loaded.

But if your TRS-80 had only ROM, it would be a very dull machine indeed. This is where the *RAM* comes in. *RAM* or *random access memory* might be called *user* memory because it is here that your programs are stored. RAM memory can be written into, read from, and modified at will from the keyboard. It is also *volatile*, which means that its contents are lost or "forgotten" as soon as the power is removed. It is for this reason that cassette and floppy disk systems are used to save the contents of RAM before turning off the computer's power.

Generally, programs that are in constant use are best stored in ROM, where they will be instantly available. Less used programs are best saved on cassette or floppy disk, where they will not occupy valuable memory space when not in use. Later, when needed, these programs can be loaded into RAM memory.

### Static and Dynamic RAM

RAM memory is further subdivided into two categories: *static* and *dynamic*. Dynamic memories work on the principle of charge storage. Tiny MOS transistors will temporarily hold this charge, and if the memory is *refreshed* periodically (every few milliseconds), they will not "forget" their information. The advantage of dynamic memory is that the storage cells can be very small, essentially one transistor for one bit of data. This in turn allows one memory chip to hold thousands of bits of information.

The *disadvantage* to dynamic memory is that the charge on these transistors must be refreshed at regular intervals or the data will be lost. This means that more complex hardware is needed.

Today, most manufacturers have opted for dynamic RAM despite the increased support hardware complexity. This is because the bit density is very high. For example, the *4116* is a dynamic RAM and eight of these chips connected in parallel will yield a *16K* memory block. The TRS-80 may be expanded to 48K by adding three such 16K blocks.

In this experiment we will be working with *static* memory chips because of the ease of interfacing. No refresh circuitry is required. The penalty we must pay is the bit density. The *2114* RAM chips that we will use are organized internally as *1K* words of *4* bits each (1K × 4). A 1K-byte memory will therefore require two of these RAM chips.

Table 7-1 lists several commonly used static and dynamic RAM chips. Note that the top-of-the-line dynamic RAMs have bit densities considerably greater than the corresponding static RAMs.

TABLE 7-1 POPULAR STATIC AND DYNAMIC RAM CHIPS

| Part number | Type | Total bits | Organized as: | Package |
|---|---|---|---|---|
| 1101 | Static | 256 | 256 × 1 | 16-pin DIP |
| 2112 | Static | 1,024 (1K) | 256 × 4 | 16-pin DIP |
| 2102 | Static | 1,024 (1K) | 1,024 × 1 | 16-pin DIP |
| 2114 | Static | 4,096 (4K) | 1,024 × 4 | 18-pin DIP |
| 1103 | Dynamic | 1,024 (1K) | 1,024 × 1 | 18-pin DIP |
| 4027 | Dynamic | 4,096 (4K) | 4,096 × 1 | 16-pin DIP |
| 4116 | Dynamic | 16,384 (16K) | 16,384 × 1 | 16-pin DIP |
| 4164 | Dynamic | 65,536 (64K) | 65,536 × 1 | 16-pin DIP |

## Hardware Interface

The logic symbol for the *2114* is shown in Fig. 7-1. Ten address lines are required (A0 through A9) to select all *1024* 4-bit words ($2^{10} = 1024$). Unless the $\overline{CS}$ (chip select) input is low, the data bus lines (D0 through D3) go to a *high-impedance* (tri-state) condition. The $\overline{WE}$ (write enable) input must be low when writing data into the chip (a memory write operation) and high when reading data out of the chip (a memory read operation). Because data flows in and out of the memory on the *same* data lines, the *2114* is said to have a *bidirectional* data bus.

A 1K memory interface to the TRS-80 is shown in Fig. 7-2. Note the following about this circuit:

1. Two *2114* chips are required to store the 8-bit words.
2. Address bus lines A0 through A9 are connected in *parallel* to the two chips. In this way the *same* 4-bit word is selected simultaneously in both memory chips.

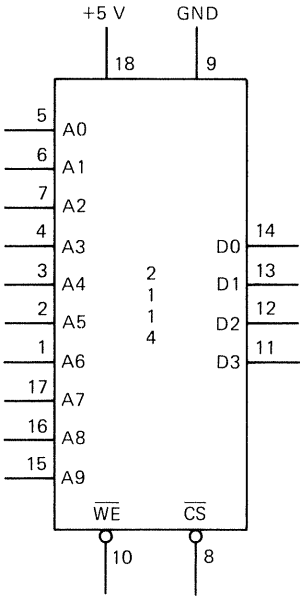
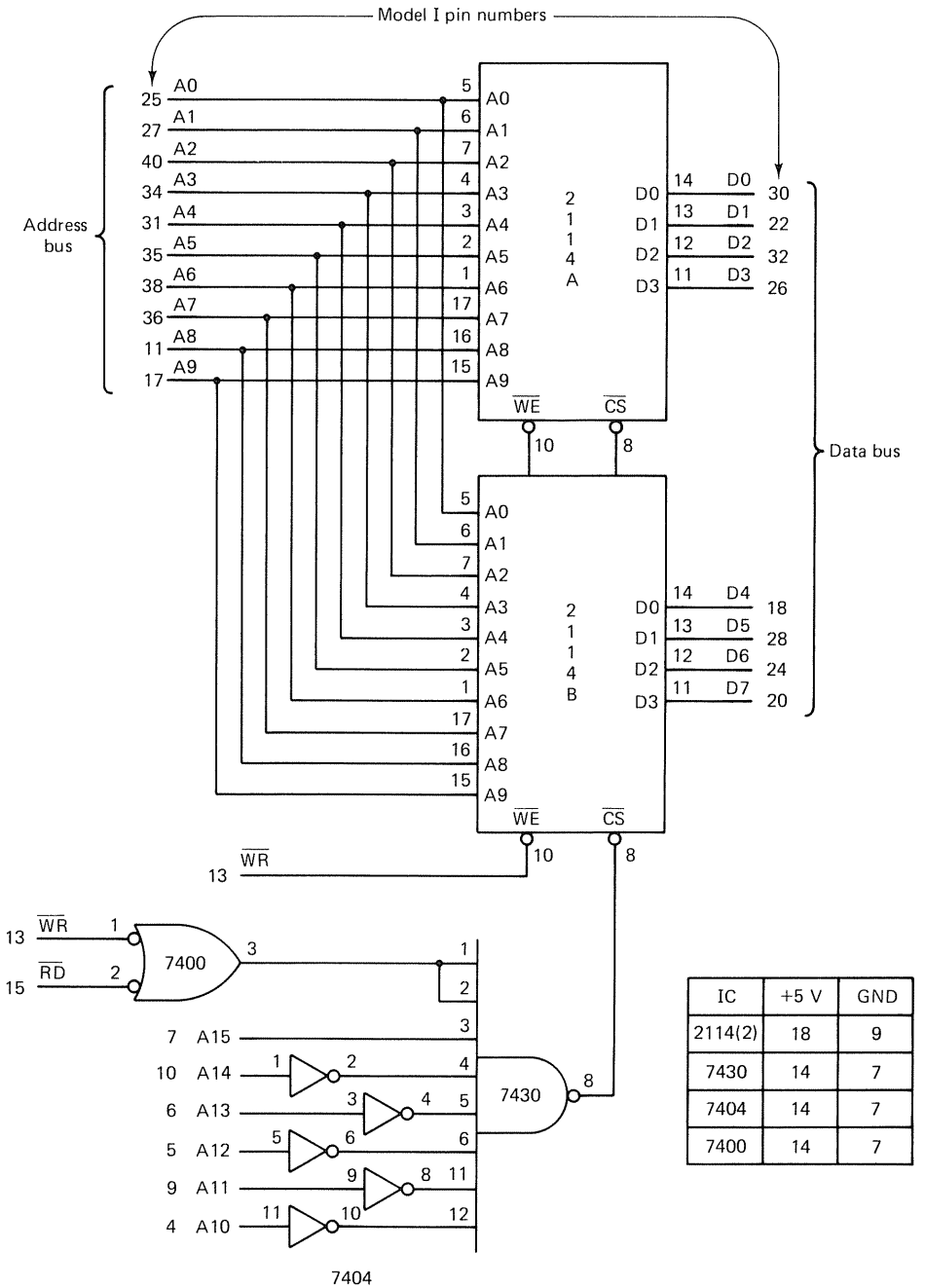



Figure 7-1 Logic symbol for the 2114 1K × 4 static RAM chip.

3. The $\overline{CS}$ inputs of both circuits are connected together and are low only if the computer is doing a *memory read* ($\overline{RD}$) or *memory write* ($\overline{WR}$) operation AND the high-order address is 100000XXXXXXXXXX (the X's represent the lines decoded by the *2114s*). This means that the 7430 output will be low for any memory read or write operation between addresses 32768 (1000000000000000) and 33791 (1000001111111111), corresponding to the 1K memory block established by the two *2114*'s.
4. The $\overline{WE}$ inputs are connected together and enabled by the $\overline{WR}$ control signal. If the address bus causes *all* 7430 inputs to be high, and a command is given that makes $\overline{WR}$ go low, any data output by the microcomputer on its data bus will be written into the memory chips at the address specified by A0 through A9.

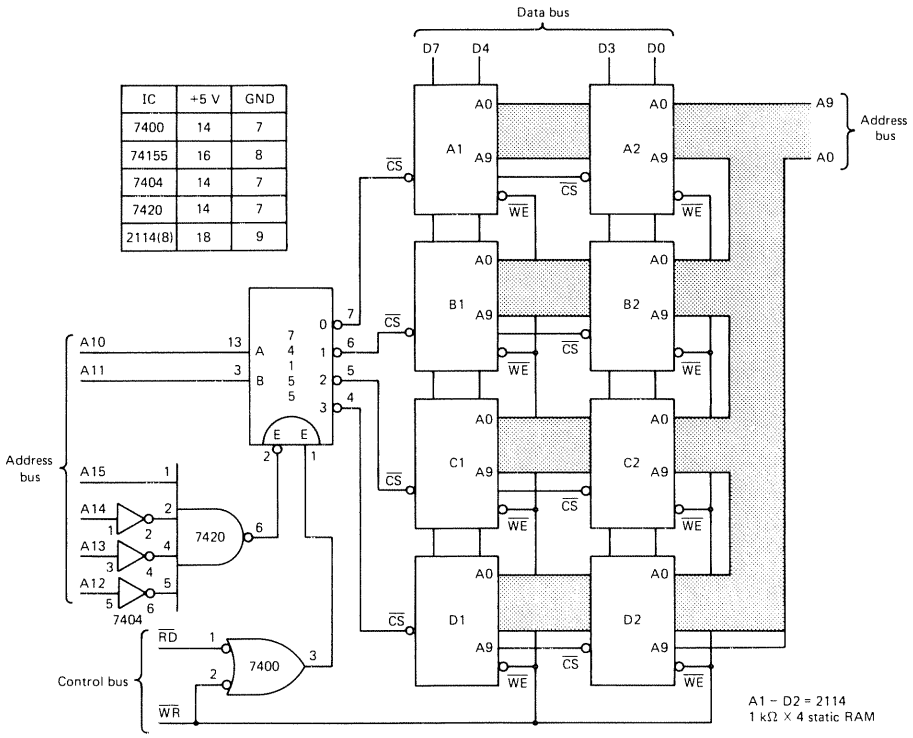
The circuit in Fig. 7-2 can be expanded to larger memory blocks by adding two more *2114*'s for each additional 1K block of memory. The address decoder is slightly more complex now because each bank of chips must be enabled separately. This is illustrated in Fig. 7-3 for a *4K* memory interface.

*Eight* RAM chips are now required and they all have their address lines connected in parallel. In addition, high-order data RAMs (A1–D1) have their data bus lines connected in parallel, as do the low-order chips (A2–D2). Although it might appear that all these parallel connections *short* each other out, this does not happen. The reason is that each bank of chips is enabled *separately* by the 74155. For example, when A10 and A11 are both low, pin




| IC | +5 V | GND |
|---|---|---|
| 2114(2) | 18 | 9 |
| 7430 | 14 | 7 |
| 7404 | 14 | 7 |
| 7400 | 14 | 7 |

Figure 7-2 A 1K memory interface to the TRS-80.




| IC | +5 V | GND |
|---|---|---|
| 7400 | 14 | 7 |
| 74155 | 16 | 8 |
| 7404 | 14 | 7 |
| 7420 | 14 | 7 |
| 2114(8) | 18 | 9 |

Figure 7-3 A 4K static memory interface. Eight 2114 RAM chips are required. Refer to Fig. 7-2 for the 2114 and TRS-80 pin numbers. The 74155 is a two- to four-line active low output decoder.

7 of the 74155 is low and *only* the A1A2 RAM bank is enabled. The other six RAM chips are disabled and their outputs are in a tri-state condition. Similarly, when A11A10 = $10_2$, pin 5 of the 74155 is low and the C1C2 bank of RAMs is enabled. Because only one bank of chips can be enabled at a time, there is no problem with one chip shorting the outputs of another.

Figure 7-4 should help explain the address decoding used in this memory interface.

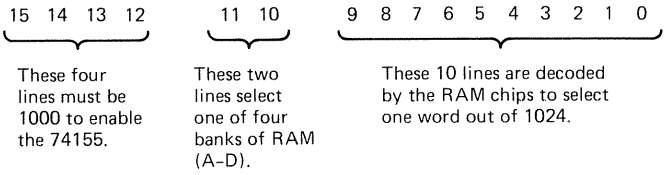



Figure 7-4 The 16 address lines of the TRS-80 are broken down into three groups for the 4K memory interface shown in Fig. 7-3.

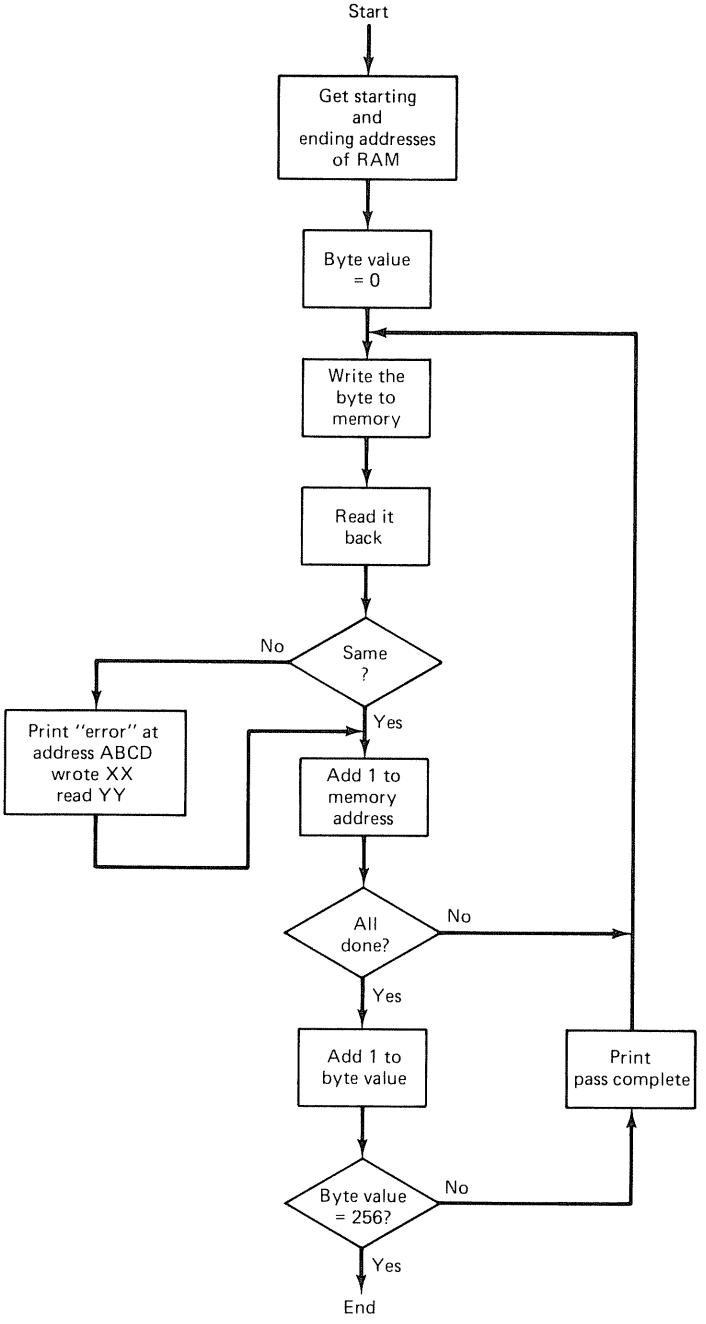



Figure 7-5 **Flowchart of a program to test a block of RAM memory.**

Although the circuit in Fig. 7-3 is straightforward, there are numerous connections to be made. And this circuit is only a *4K* memory! To achieve 16K, *32* RAM chips are needed! This is why most manufacturers use the higher-density dynamic RAM chips such as the *4116*.

### Testing RAM Memory

Once we have installed our new block of memory, how can we be sure that it is operational? Probably the best technique is to test it with *software*. Figure 7-5 illustrates a possible flowchart to do this. First the beginning and ending addresses are obtained, then all possible binary combinations are written to and read back from each location in this block. If no errors occur, the program will take no further action. If an error does occur, its location, the value written, and the value read back are all displayed on the screen. Note that the program does not quit at this point, but continues testing. A BASIC program solution for this flowchart is included in the "Procedure" section.

More elaborate test programs could actually identify the faulty RAM chip by *number* to make troubleshooting all the easier. At any rate, a memory test program is a good one to run on your TRS-80 from time to time to make sure that your computer is functioning properly.

## PROCEDURE

**Step 1.** Refer to Fig. 7-2 and carefully wire this circuit on your breadboard. Take care when handling the *2114* RAM chips, as they are subject to damage by *static discharge*.

**Question 7-1.** What BASIC command is used to write to the *first* memory location in this block of RAM?

**Question 7-2.** What BASIC command is needed to read data from the *last* memory location in this block of RAM?

**Step 2.** Test your hardware by running the following program.

```
10 CLS
20 INPUT "DATA TO BE WRITTEN IS: ";D
30 POKE-32768,D
40 Y=PEEK(-32768)
50 PRINT "DATA READ BACK IS: ";Y
60 GOTO 20
```

**Step 3.** Refer to the flowchart in Fig. 7-5 and write a corresponding memory test program in BASIC. Using this test program, test the complete 1K block of RAM. A solution is provided at the end of the chapter.

**Step 4.** Remove power from your breadboard and then pull out one of the RAM chips. Restore power and run the test program.

**Question 7-3.** Explain the results in step 4.

**Step 5.** Reread the objectives as stated at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

## SOLUTIONS TO QUESTIONS

**7-1.** POKE-32768,Y. Recall that all addresses greater than 32767 must be *calculated* as address = desired address - 65536.

**7-2.** Y=PEEK(-31745).

STEP 3 (solution) This is one possible solution. Because the program is written in BASIC, it is *very slow*. To test 1K of memory will require nearly *2 hours*! However, each pass does test *each* memory location. With machine language, much faster versions can be written.

```
  5 REM TRS-80 RAM TEST PROGRAM
 10 CLS
 20 INPUT "PLEASE ENTER THE STARTING
    ADDRESS OF THE TEST ";S
 30 INPUT "NOW ENTER THE ENDING
    ADDRESS ";E
 40 FOR B=0 TO 255                          :REM B=BYTE VALUE
 50 FOR J=S TO E
 60 IF J>32767 THEN J1=J-65536 ELSE J1=J
 70 POKE J1,B
 80 IF PEEK(J1)=B THEN 100
 90 PRINT "ERROR AT ";J;" WROTE ";B;
    "READ ";PEEK(J1)
100 NEXT J
110 PRINT "PASS ";B;" COMPLETE"
120 NEXT B
130 END
```

**7-3.** With one RAM chip missing, four of the data lines will be *open*. These lines will probably be interpreted by the TRS-80 as logic 1's. In this case, when the first word (00000000) is written to RAM, it will be read back as 15 (00001111) or 240 (11110000), depending on which RAM chip is missing.

# 3 Special Interfacing Problems

The experiments in this section should not be attempted until the concepts in Part 2 have been mastered. This section covers the special problems that occur when interfacing the TRS-80 to the outside world.

# EXPERIMENT 8

## HARDWARE INTERFACING TECHNIQUES, PART 1: INPUTS

### OVERVIEW

In this experiment you will interface several different input sensors to the TRS-80. These include a *magnetic switch*, a *temperature sensor*, and a *photocell*. All circuits will use an 8255 in an *I/O-mapped I/O* configuration. Control programs using BASIC will be developed.

### OBJECTIVES

The key points to be learned from this experiment are:

1. The outside world generally presents data to the microcomputer in *non-TTL*-compatible voltage levels.
2. The TRS-80 can only interpret *TTL* levels, and others may damage the computer.
3. An *analog comparator* circuit is commonly used to convert nonstandard voltages to TTL.
4. One 8-bit input port allows eight different sensors to be monitored by the computer.

### PARTS LIST

1 7404 hex inverter
1 7430 8-input NAND gate
1 8255 programmable peripheral interface (PPI) (JAMECO INS 8255)
1 LM339 quad comparator (Radio Shack 276-1712)

1 LM334 temperature sensor (Radio Shack 276-1734)
1 cadium sulfide photoresistor (Radio Shack 276-116)
1 magnetic switch (Radio Shack 49-495)
4 1-kΩ resistors (brown-black-red)
2 10-kΩ resistors (brown-black-orange)
1 220-Ω resistor (red-red-brown)
2 10-kΩ variable resistors

## DISCUSSION

### Inputting Data from the Outside World

The true power of the microcomputer becomes clear when we begin interfacing it to the *outside world*. Imagine being able to turn on or off any light or appliance within your home. Or sensing the temperature in your living room and activating the furnace or air conditioner. The possibilities are almost endless. The main obstacle to doing any of these things is the electronics required between the computer and the controlled device.

Figure 8-1 illustrates the problem. The microcomputer is a binary machine, communicating only in 1's and 0's. Unfortunately, most information we wish to monitor is *not* in this digital format. Consider temperature for example. The temperature could be 10° or 80° or any value in between. Yet the computer expects a simple *yes* or *no* type of input. Quantities such as temperature, pressure, humidity, and velocity are all *analog* in nature and cannot simply be described as being *ON* or *OFF*. Thses types of inputs can

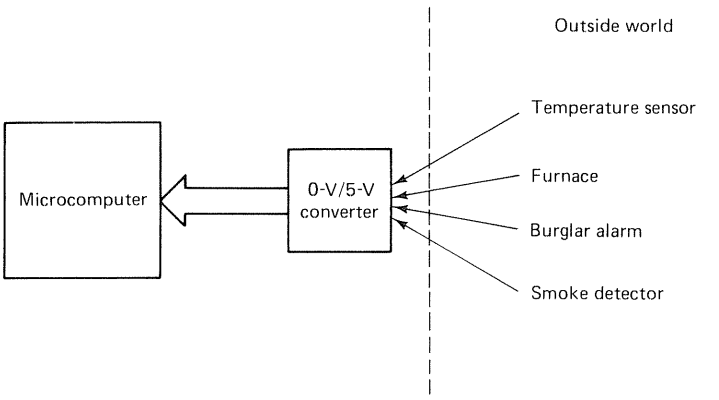



Figure 8-1 The outside world is interfaced to the microcomputer through a circuit which must convert the sensor's information to 0-V and 5-V levels, compatible with the computer's electronics.

be interfaced to the computer, but they require a special *analog-to-digital converter* circuit. This technique is examined in detail in Experiment 11.

Fortunately, much of the information we might want our computer to acquire is of a yes-no nature. For example, is the temperature 60°, or is the humidity less than 50%, or is the furnace off? All these questions can be answered yes or no. The problem now becomes one of converting the sensor's *ON-OFF* status to 0-V and 5-V logic levels that the computer can use. This is the 0 V/5 V converter box in Fig. 8-1.

The next few sections present some examples of the circuitry required to convert the *ON-OFF* information of the sensor to standard logic levels.

## Magnet Switch Interface

One of the most common and simplest devices to interface to the computer is a *mechanical switch*. This device certainly meets our criteria of having only an *ON* or *OFF* condition. As you read through this example, note how the open or closed switch contacts are converted to standard logic levels.

Magnetic switches are commonly used in burglar alarm systems to detect door or window openings. Two types of switch are possible. In one, the switch contacts *close* when the magnet engages; in the other, the switch contacts *open* when the magnet engages. Figure 8-2 shows an example of the former type. The switch itself is housed in the plastic package shown on the right in the photograph, while the leftmost package is simply a magnet. An advantage of this type of switch is that the switch contacts are not exposed to the environment, therefore allowing reliable contact closures even under very dirty conditions.

Figure 8-3 illustrates an interface to the TRS-80 using the 8255 PPI chip. The magnetic switches shown are being used to detect door openings and the contacts *close* when the magnets are engaged. With the door closed, the 7404 input is held at ground (0 V) and the 8255 inputs (PC0 and PC1)

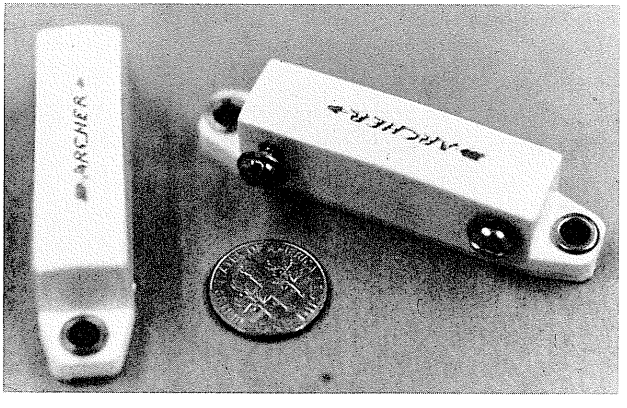
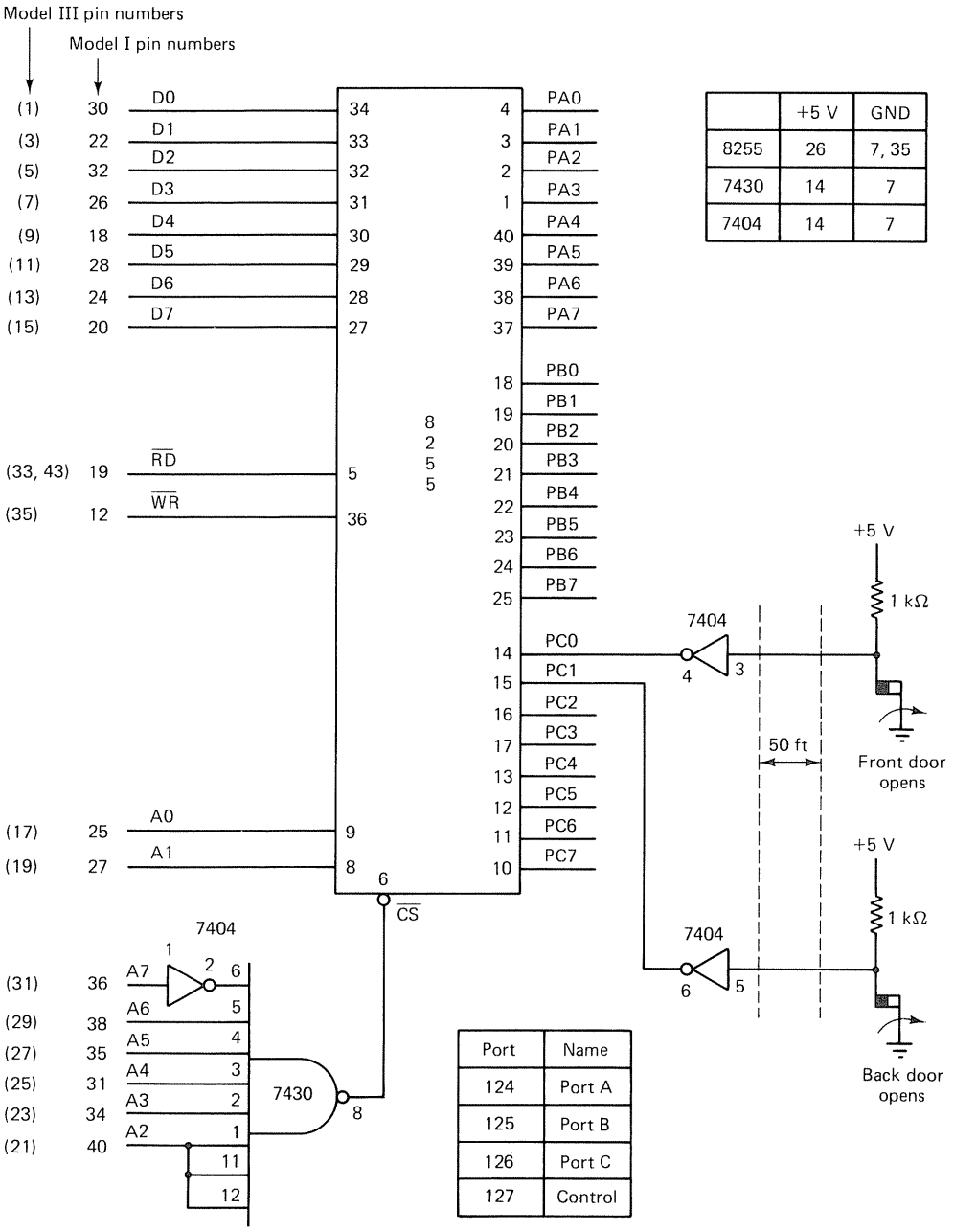



Figure 8-2 Typical magnetic switch. When the two units are brought within an inch of each other, the switch contacts will close. The unit shown is a Radio Shack 49-495.




| | +5 V | GND |
|---|---|---|
| 8255 | 26 | 7, 35 |
| 7430 | 14 | 7 |
| 7404 | 14 | 7 |

| Port | Name |
|---|---|
| 124 | Port A |
| 125 | Port B |
| 126 | Port C |
| 127 | Control |

Figure 8-3 Two magnetic switches are used to monitor the front and back doors of a home. When either door opens, the 7404 inverter input goes high and the input port receives a low level.

are high. An open door disengages the magnets, opening the contacts and causing the inverter input to go high. The 8255 input is now low. A BASIC program to check the two doors might be:

```
 5 CLS
10 OUT 236,16                    :REM MODEL III ONLY
15 OUT 127,137                   :REM INIT PPI. A,B=OUTPUTS, C=INPUT
20 IF (INP(126) AND 1) <>1 THEN
   PRINT "FRONT DOOR OPEN"
30 IF (INP(126) AND 2) <>2 THEN
   PRINT "BACK DOOR OPEN"
40 END
```

Note that lines 20 and 30 use the *masking* technique for testing individual bits described in Experiment 4. Using the 8255 chip, *24* inputs can be monitored. The wiring from each switch to the computer will require a pair of conductors (twisted pair), but the 7404 and pullup resistor may be located at the computer site.

As discussed previously, many quantities in the outside world are *analog* in nature and do not exist in a simple *ON-OFF* condition. As you read the next two examples, note how the *analog* information is converted to a *digital* (1 or 0) format.

### Sensing Light and Dark

There are numerous instances where it would be desirable for your computer to sense the relative amount of light in a room or out-of-doors. For example, an outdoor sensor could detect nightfall and notify the computer, which could in turn activate a yard light. In other cases it might be desirable to determine if the light in a room has been switched on (or off).

A useful integrated circuit for accomplishing this type of interface is the *LM339 analog quad comparator* shown in Fig. 8-4. This chip has four (quad) identical comparators in one 14-pin package. Each comparator has two inputs, labeled (+) and (-). Electronically, the comparator compares the voltage on its two inputs and if the (+) input is largest, the output goes to an *open* circuit (but can be pulled high by an external resistor). If the (-) input is the largest, the output switches to 0 V. A particularly nice feature of the LM339 is that only a few *microvolts* of difference between the two inputs are required for the output to switch states, and this switching typically occurs in less than 1 $\mu$s.

The LM339 is also useful for converting voltages that exceed normal TTL values to standard levels. This is because the comparator inputs are rated to withstand voltages as high as +36 V (the minimum is -0.3 V).

A simple circuit for detecting the presence of light and converting this information to standard logic levels is shown in Fig. 8-5. This circuit uses

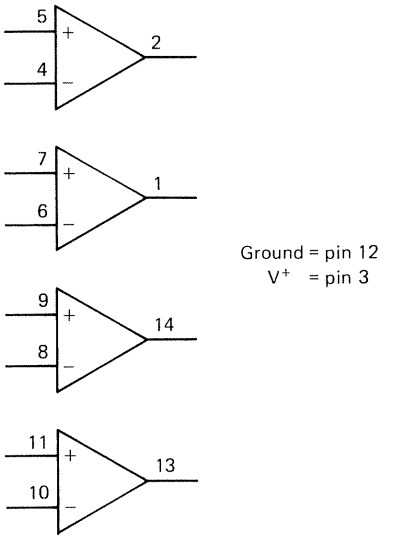



Figure 8-4 LM339 quad comparator. Four comparators are included in one standard 14-pin dual-in-line (DIP) package. The output switches between an *open circuit* when the (+) input is greater than the (−) input, and *0 V* when the (−) input is greater than the (+) input.

a *photoresistor* (shown in Fig. 8-6) and one-fourth of the LM339 quad comparator. When the comparator (+) input (pin 7) has a more positive voltage than the (−) input (pin 6), the output of the comparator goes *OFF* and is pulled to +5 V by R3. When the (−) input is most positive, the output is at 0 V. Because the photoresistor's resistance varies from *hundreds* of ohms in bright light to *megohms* in darkness, the voltage at pin 7 will vary from 0 V (bright) to 5 V (dark). The circuit can be adjusted by setting the light conditions to the switching point and then adjusting R1 so that the output

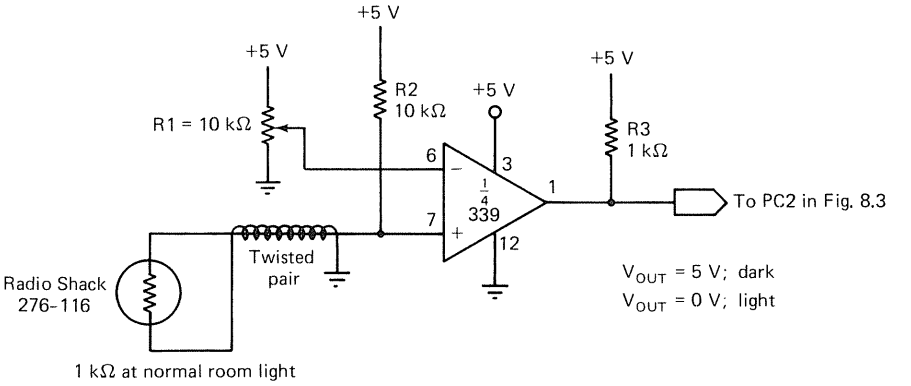



Figure 8-5 Light-sensing circuit with adjustable threshold. This circuit uses one of the four comparators in the LM339 package. The photoresistor has a resistance which is low (~100 Ω) in the presence of light but large (~1 MΩ) in darkness. By adjusting R1, the specific light-dark switching point can be set.

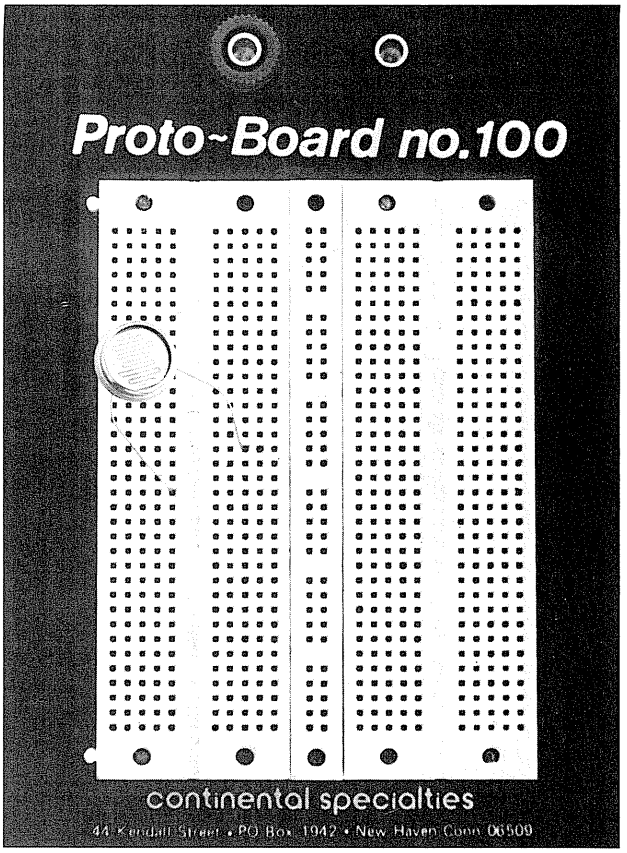



Figure 8-6 Typical light sensor. The unit shown is a cadmium sulfide photoresistor (Radio Shack 276-116).

(pin 1) just switches to the desired value (5 V for dark or 0 V for light). A BASIC test program for aligning the interface might be (assuming that PC2 in Fig. 8-3 is the input bit)

```
 5 CLS
10 OUT 236,16                                  :REM MODEL III ONLY
15 OUT 127,137                                 :REM INIT PPI
20 IF (INPUT(126) AND 4)=4 THEN PRINT@480,
   "DARK " ELSE PRINT@480, "LIGHT"
30 GOTO 20
```

Again a twisted pair can be used to connect the photoresistor to the interface electronics. With four comparators in one package, four such sensors could be monitored.

One application of the photoresistor interface that should not be overlooked is a *touch sensor* input to the computer. When action is to occur, the user simply covers the photoresistor with a finger. Because the sensor can be

mounted behind a plate of glass, this technique is useful when it is desired for security reasons to limit access to the computer.

## Sensing Temperature

Temperature is another analog quantity that can be monitored by the TRS-80. In this particular application we are not interested in the specific temperature in degrees, but rather in the ability to determine if the temperature is above or below some *trigger* value.

Figure 8-7 illustrates the electrical interface. An *LM334 temperature sensor* is used to provide a *current* that is proportional to temperature. With the component values shown, the voltage applied to the comparator positive input will increase at a rate of *10 mV/°K*. For example, with $V^+ = 5$ V and the components shown, the output voltage ($V^-$) is typically about 3.1 V at room temperature. However, this voltage will rise as the temperature increases. By adjusting R3 to a particular value, we may detect when the temperature exceeds this limit.

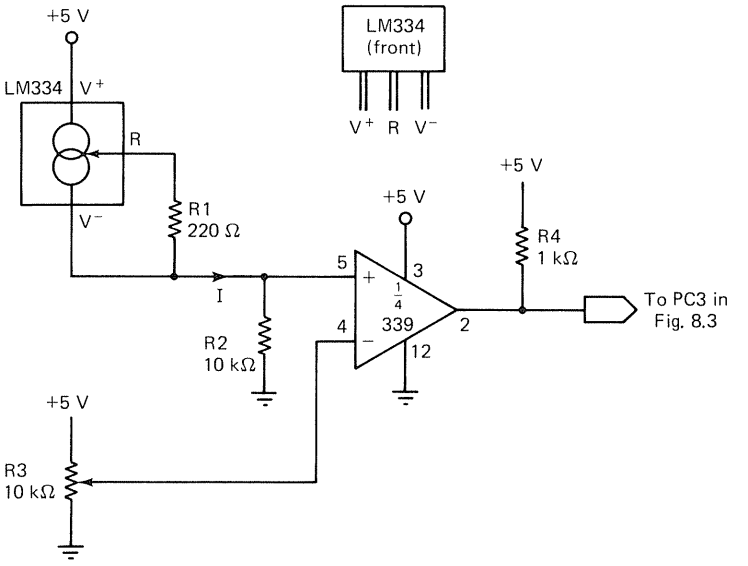



Figure 8-7 Sensing temperature with the TRS-80. The LM334 output current increases with temperature such that $V_{R2}$ will rise 10 mV/°K. When the LM339 comparator (+) input (pin 5) exceeds the trigger voltage established by R3 on pin 4, its output pulls high through R4. For temperatures below this point, the output is low.

Example 8-1

Refer to Fig. 8-7. Assume that the room temperature is 20°C (68°F) and $V^-$ (LM334) = 3.1 V. Using R3, what voltage should be applied to pin 4 of the comparator if we wish to detect temperatures above 32°C (90°F)?

**Solution** The change in temperature is 32°C - 20°C = 12°C. Because a Kelvin degree is the same as a Celsius degree, $V^-$ will increase by 12° × 10 mV/°K = 120 mV. R3 should be set so that pin 4 is 3.1 V + 0.12 V = 3.22 V.

The LM334 temperature sensor is contained in a TO-92 plastic package and looks identical to a small plastic transistor. The 220-Ω resistor (R1) can be soldered directly to the two device leads and the remaining two connections ($V^+$ and $V^-$) brought to the interface electronics through a two-conductor cable or twisted pair. Because the device is a *current source*, the resistance of the cable, and therefore its length, is not important.

The best way to calibrate the circuit in Fig. 8-7 would be to place the LM334 at the temperature condition you wish to detect and then measure the output voltage across R2. Now R3 can be adjusted until the voltage at pin 4 just equals this value. If you do not have a voltmeter, the following program can be run. With the LM334 at the trigger temperature, slowly adjust R3 until the screen displays "HOT." This program assumes that the 339 output is connected to PC3 in Fig. 8-3.

```
 5 CLS
10 OUT 236,16                                  :REM MODEL III ONLY
20 IF (INP(126) AND 8)=8 THEN PRINT@480,
   "HOT " ELSE PRINT@480, "COLD"
30 GOTO 20
```

### Smoke Detector Interface

Occasionally, the signal we wish to monitor is not steady but of a *pulsing* nature: for example, the alarm on a clock or smoke detector. In this case, the interface circuitry must *latch* the alarm condition to prevent the computer from assuming multiple alarms. This generally also requires some means of *resetting* the latch once the alarm has ceased.

In this final example let us see how to interface a smoke detector to the TRS-80. Figure 8-8 illustrates a common type. This is a Radio Shack ionization smoke alarm and it produces a burst of 3 kHz audio at a rate of 4 to 6 Hz when activated.

The interfacing scheme consists of simply paralleling wires with the smoke detector's internal audio alarm. In the case of the Radio Shack unit this involves prying the front cover off (voiding the warranty) and connecting two wires to the outermost connections of the audio alarm.

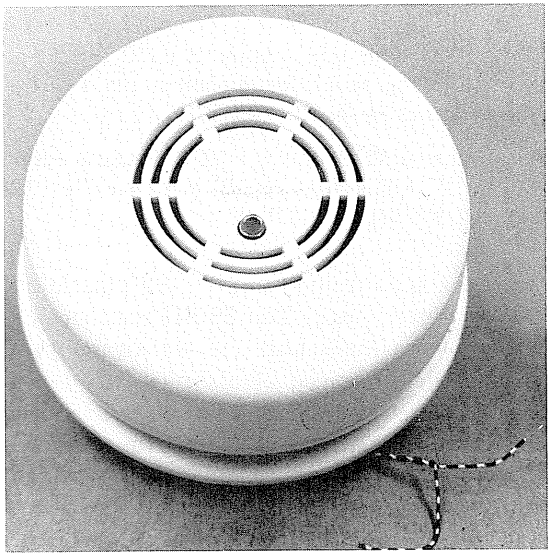


Figure 8-8 Smoke detector and alarm. Illustrated is the Radio Shack ionization smoke alarm 49-454. The two wires shown are in parallel with the internal speaker.

Because the alarm is pulsing at 4 to 6 Hz, the interface circuit must use a flip-flop to hold the alarm input to the computer at a steady (nonpulsing) logic level. The interface circuit is illustrated in Fig. 8-9. When the smoke alarm is activated, the pulsing signal is applied as a clock to the 7476 JK flip-flop. With J wired high and K low, the Q output will switch *high* on the first pulse from the alarm. This can be monitored by the computer for fur-

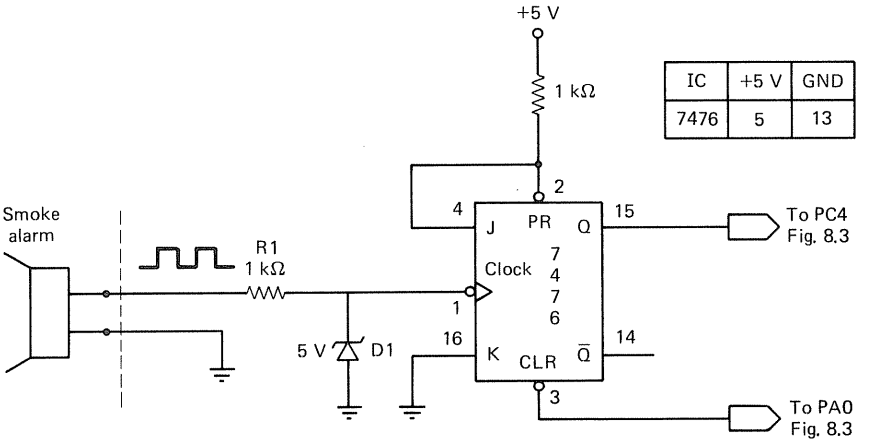



Figure 8-9 TRS-80 smoke detector interface. PC4 and PA0 refer to the 8255 schematic diagram in Fig. 8-3. The flip-flop is necessary because the alarm signal is *pulsing*. D1 and R1 protect the input from large or negative input voltages.

ther action (for example, an autodial telephone interface to call the fire department).

Because the flip-flop output will remain high even after the alarm disappears, the clear input must be pulsed low to reset the circuit. This is accomplished by pulsing the PA0 line.

The following program will clear the flip-flop (in case it powers up already set) and then wait for an alarm condition.

```
 5 CLS
10 OUT 236,16                    :REM MODEL III ONLY
15 OUT 127,137                   :REM INIT PPI
20 OUT 124,0: OUT 124,1:         :REM PA0 RESETS THE FLIP-FLOP
30 IF (INP(126) AND 16)=16 THEN
   PRINT@480, "ALARM" ELSE 30
40 END
```

PROCEDURE

**Step 1.** Study the circuit shown in Fig. 8-3 and wire the 8255 PPI, 7430, and 7404 on your breadboard. For the moment connect nothing else to the 8255.

**Question 8-1.** We can test that the hardware is functional at this point by programming all three PPI ports to be *inputs* and selectively grounding one input at a time while holding all others high. What are the eight different input combinations that BASIC will see as each pin is grounded?

**Question 8-2.** What command should be used to initialize the PPI?

**Step 2.** Load and run the following program, testing all three ports by touching one input pin to ground at a time, with the other seven connected to +5 V. Do not proceed until all ports check out.

```
 5 CLS
10 OUT 236,16                    :REM MODEL III ONLY
15 OUT 127,155                   :REM PROGRAM ALL PORTS AS
                                  INPUTS
20 INPUT "WHICH PORT DO YOU WISH
   TO TEST (A, B OR C) ";P$
30 IF P$="A" THEN N=124 ELSE IF
   P$="B" THEN N=125 ELSE N=126
40 PRINT@480, INP(N)
50 GOTO 40
```

**Step 3.** Connect two wires to the contacts on one magnetic switch. Connect one of these wires to ground and the other wire to pin 3 of the

7404 using a 1-kΩ pullup resistor as shown in Fig. 8-3. Connect the 7404 output (pin 4) to the PC0 input (pin 14) of the 8255.

**Step 4.** Using the program given in the discussion as a guide, write a similar program to detect when the magnets are close together ("closed") or far apart ("open"). Do not forget to initialize the PPI.

**Step 5.** Now interface the light-sensing circuit shown in Fig. 8-5. Connect the LM339 output (pin 1) to PC2 (pin 16) of the 8255. Load and run the test program given in the discussion. Play with the setting of R1 and see if you can get the computer to display "LIGHT" and "DARK" as the room lights are turned on and off.

**Step 6.** Using another comparator in the LM339, construct the temperature-sensing circuit shown in Fig. 8-7. Connect the comparator output (pin 2) to PC3 (pin 17) of the 8255. Now load and run the program given in the discussion.

**Step 7.** Adjust R3 until the screen just displays "COLD." Now pinch the LM334 between your fingers. In 5 to 10 s, the screen should display "HOT."

*Note.* All three sensors should now be interfaced and adjusted. In the following step we will develop a control program to monitor particular conditions of *all three* sensors.

**Step 8.** Write a BASIC program that:

1. Displays the status of all three sensors on the screen.
2. Asks for the "alarm condition" that should cause the computer to display or flash "ALARM." For example, if the door is *open*, the temperature is *hot*, and the light is *dark*, sound the alarm.

One possible solution is provided at the end of this experiment.

**Step 9.** Reread the objectives as stated at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

*Note.* Do not disassemble the hardware for this experiment, as portions of it will be used in Experiment 9.

## SOLUTIONS TO QUESTIONS

8-1. When the least significant bit is grounded, the input port will see 11111110 or $254_{10}$. The seven other combinations are: 253, 251, 247, 239, 223, 191, and 127.

8-2. OUT 127,155.

STEP 8 (solution):

```
  5 CLS
 10 OUT 236,16                          :REM MODEL III ONLY
 15 OUT 127,137                         :REM INIT PPI A,B=OUTPUTS,
                                         C=INPUT
 20 PRINT "WHAT IS THE ALARM
    CONDITION?"
 30 PRINT
 40 INPUT "DOOR OPEN OR CLOSED ";
    D$
 50 IF D$="OPEN" THEN D=0 ELSE IF
    D$="CLOSED" THEN D=1 ELSE 40
 60 INPUT "LIGHT CONDITION:
    BRIGHT OR DARK ";L$
 70 IF L$="BRIGHT" THEN L=0 ELSE
    IF L$="DARK" THEN L=1 ELSE 60
 80 INPUT "TEMPERATURE
    CONDITION: HOT OR COLD ";T$
 90 IF T$="HOT" THEN T=1 ELSE IF
    T$="COLD" THEN T=0 ELSE 80
100 CLS
110 OUT 236,16                          :REM MODEL III ONLY
115 P=INP(126)                          :REM SAMPLE THE SENSORS
                                         FROM PORT C
120 IF (P AND 1)=1 THEN
    D$="CLOSED" ELSE D$="OPEN  "
130 IF (P AND 4)=4 THEN L$="DARK  "
    ELSE L$="BRIGHT"
140 IF (P AND 8)=8 THEN T$="HOT "
    ELSE T$="COLD"
150 PRINT@20, "THE DOOR IS ";D$
160 PRINT@148, "THE LIGHT IS ";L$
170 PRINT@276, "THE TEMPERATURE
    IS ";T$
180 A=D+4*L+8*T                         :REM THIS IS THE VALUE OF
                                         THE ALARM CONDITION
190 P=(P AND 13)                        :REM 13 IS A MASK TO FORCE
                                         ALL UNUSED BITS TO 0
200 IF P=A THEN PRINT@468,
    "A L A R M" ELSE PRINT@468,
     "          "
210 GOTO 115                            :REM CYCLE, MONITORING FOR
                                         ANY CHANGES
```

*Note.* If you wish to *simulate* this program without building the hardware, replace line 115 with 115 P=RND(255).

# EXPERIMENT 9

## —HARDWARE INTERFACING TECHNIQUES, PART 2: OUTPUTS—

### OVERVIEW

In this experiment you will wire an *npn* transistor as a mechanical relay driver. A *solid-state relay* will also be interfaced and driven directly from TTL. Finally, an *ultrasonic* interface will be constructed for controlling 120-V AC devices. Control programs using BASIC will be developed.

### OBJECTIVES

The key points to be learned from this experiment are:

1. Few, if any, real-world devices can be powered or driven directly by a microcomputer.
2. A mechanical relay can be used to control most electronic and electromechanical devices. It can be driven by a microcomputer via an output port and external *driver* transistor.
3. *Solid-state* relays can be driven directly from TTL but may not handle the current and voltage of their mechanical counterparts.
4. When *isolation* between the controlled device and the computer is desired, *opto-couplers* and *ultrasonic* techniques can be used.

### PARTS LIST

1 7404 hex inverter
1 7430 8-input NAND gate
1 8255 programmable peripheral interface (PPI) (JAMECO INS 8255)

1 7400 quad NAND gate
1 LM386 integrated amplifier (Radio Shack 276-1731)
1 LM567 tone decoder (Radio Shack 276-1721)
1 TIL111 opto-coupler (Radio Shack 276-132)
1 magnetic switch (Radio Shack 49-495)
1 piezoelectric buzzer (Radio Shack 273-060)
1 SPDT DIP relay (Radio Shack 275-216)
1 general-purpose diode
1 general-purpose *npn* transistor
2 40-kHz transducers (Micromint Inc., 917 Midway, Woodmere, NY 11598)
1 LED
1 100-Ω resistor (brown–black–brown)
1 220-Ω resistor (red–red–brown)
1 330-Ω resistor (orange–orange–brown)
2 1-kΩ resistors (brown–black–red)
1 0.05-μF capacitor
1 0.1-μF capacitor
1 0.033-μF capacitor
1 0.005-μF capacitor
1 0.0047-μF capacitor
1 10-μF capacitor
1 100-μF capacitor
2 10-kΩ potentiometers

## DISCUSSION

### The Microcomputer as a Controller

Programming takes on new meaning when your BASIC program can turn on the light across the room as darkness approaches, or activate an audio alarm when a magnetic switch is interrupted. And these are just a few of the many possibilities.

In the last experiment we explored the *input* side of this story by using the TRS-80 to monitor the "outside world." In that experiment we were able to detect the *ON/OFF* status of various input sensors.

In this experiment we examine microcomputer *outputs*. The output devices our computer can control may be of an *analog* or a *digital* nature. For example, to generate computerized speech the computer outputs a binary number, which is then converted to a voltage level. This process is

repeated over and over, simulating the complex voltage waveform representative of human speech. A similar approach can be used to generate computer music.

Both of these examples require special *digital-to-analog* converter circuits. This type of interface is discussed in Experiment 10.

In this experiment we interface several output devices that the computer can control with simple *ON* and *OFF* commands. These include lights, relays, home appliances, sprinklers, and most any other device whose control function can be switched *ON* or *OFF*.

The main obstacle in accomplishing these interfaces is again the *control electronics* between the computer and the outside world, as illustrated in Fig. 9-1.

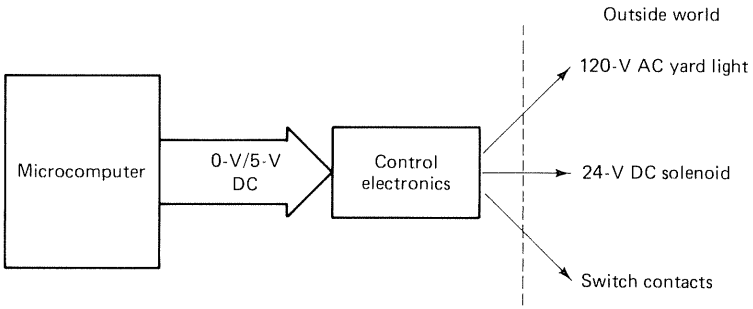



Figure 9-1 A microcomputer can be used to control any number and type of "outside-world" devices provided that the proper control electronics is used between the computer and the outside world.

The control electronics is necessary because the typical microcomputer is a 5-V DC machine, whereas the typical outside-world device is anything but 5 V DC! It might be a 24-V DC solenoid on a sprinkler system, a pair of switching contacts on a slide projector, or a 120-V AC yard light.

In addition to their non-TTL compatibility, these outside-world devices can also present a *hazard* to your microcomputer. (If you have ever seen a digital circuit board that "accidentally" encountered 120 V AC, you know what I mean!) Therefore, for some devices, another requirement for the control electronics is *isolation* (no direct electrical connection) between the computer and control circuitry.

## Mechanical Relays

Mechanical relays have been used for many years to allow *low*-voltage DC control circuits to control *high*-voltage and high-current AC or DC devices. The basic technique is illustrated in Fig. 9-2a. When S1 is closed, current is allowed to flow through the relay coil to ground, causing it to become an

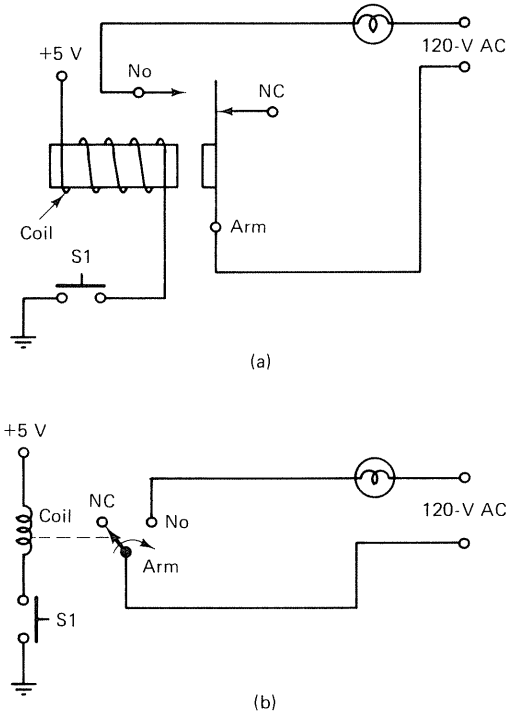
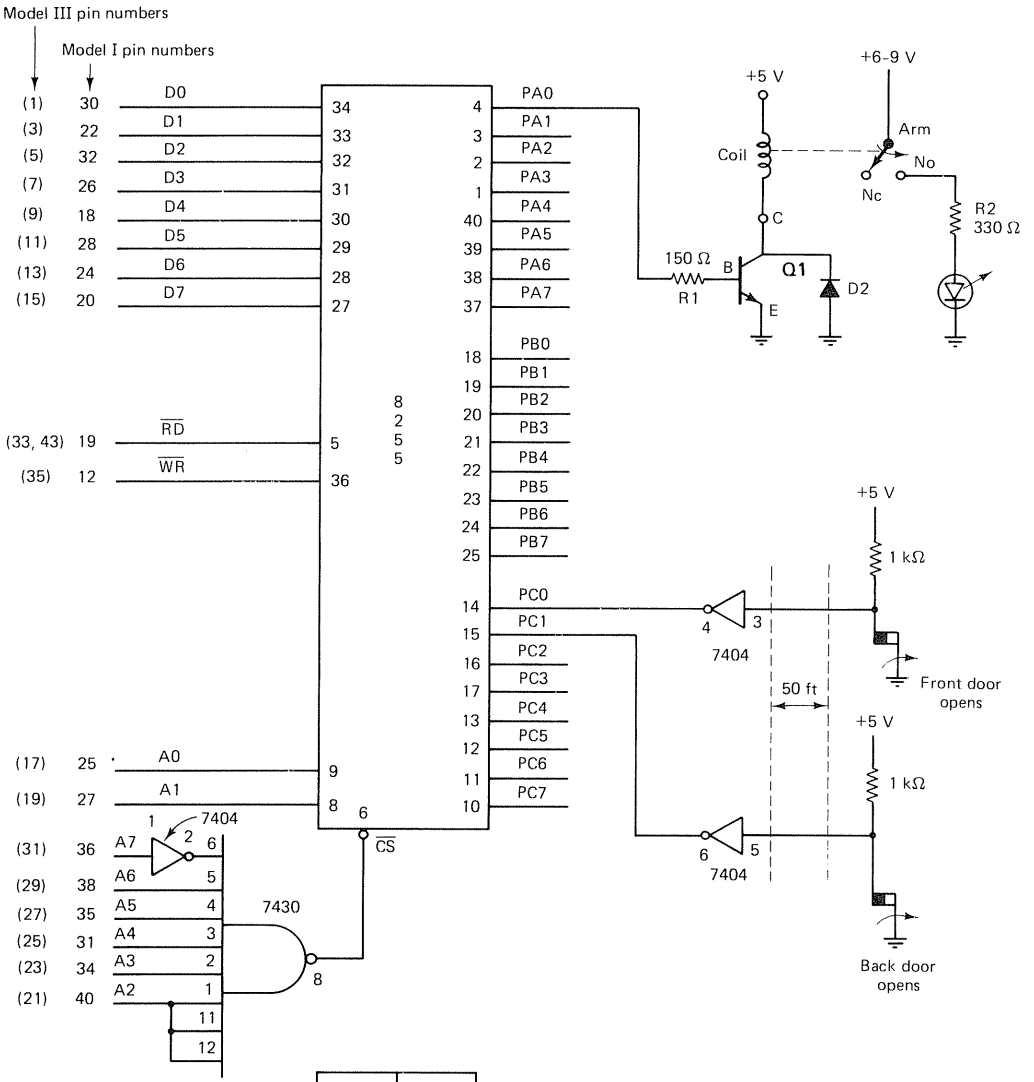



Figure 9-2 Typical relay control circuit. The circuit in (a) presents a pictorial view of the relay; the symbol used in (b) is more common on schematic diagrams. In either case, depressing S1 closes the relay contacts and activates the AC light.

*electromagnet*. This in turn pulls in the moving metal arm of the relay, causing the lamp to light as the AC current path through the *normally open* (NO) contact and *arm* is completed. Releasing the switch turns off the light. Note that the *control* circuit consists of S1, the 5-V DC source, and the relay coil. Depending on the relay type, the control current involved is usually quite small. On the other hand, the *controlled* circuit consists of the relay contacts, the light, and the 120-V AC source. The current capabilities of this circuit are determined by the size of the relay contacts. Typical values for a small 5-V relay are 1 to 3 A. Larger relays handle correspondingly higher currents.

Often, the schematic symbol for the relay is separated into a *coil* and *contact* set linked by a dashed line as shown in Fig. 9-2b.

Figure 9-3 illustrates an interface between the TRS-80 and a small 5-V DC relay. The relay is being used to control an LED powered by a 6 to 9-V DC source. This is an example of how *non-TTL* voltages can be interfaced to the microcomputer. An 8255 programmable peripheral interface chip is used and port A is programmed as an output port. Transistor Q1 takes the place of the pushbutton switch in Fig. 9-2. When PA0 goes high, this voltage will turn on the transistor, causing a short circuit to exist between its collector (C) and emitter (E) terminals. This is analogous to closing




| IC | +5 V | GND |
|---|---|---|
| 8255 | 26 | 7, 35 |
| 7430 | 14 | 7 |
| 7404 | 14 | 7 |

| Port | Name |
|---|---|
| 124 | Port A |
| 125 | Port B |
| 126 | Port C |
| 127 | Control |

Figure 9-3 Interfacing a mechanical relay to the TRS-80. The 8255 provides the output port and Q1 functions as a relay driver. Programming a 1 to PA0 will turn the relay *ON*, activating the LED. The relay shown is a subminiature single-pole-double-throw (SPDT) DIP relay (Radio Shack 275-216).

S1 in Fig. 9-2. Current can now flow through the relay coil, closing the switch contacts and lighting the LED.

You might wonder why the relay could not be connected *directly* to the 8255 output (programming PA0 to be a 0 could then turn on the relay). The problem is that the relay coil current, although not large, is still too large to be handled by the 8255 or most any TTL circuits alone. For the relay shown in Fig. 9-3, the manufacturer indicates a coil resistance of 56 Ω, which will draw about 90 mA (5 V/56 Ω) of current when energized. The 8255 can only sink *1.6 mA* and most TTL gates can handle *16 mA* maximum. The transistor, however, can easily handle this current.

When PA0 switches low, the transistor will turn off and become an *open circuit* between its collector and emitter terminals. This will turn the relay off. It will also generate a momentary, but large *inductive "kickback" voltage* at the collector terminal (current interrupted while flowing through an inductor). Diode D2 protects the transistor from damage due to this voltage transient by shorting this voltage to ground. Resistors R1 and R2 protect the transistor and LED junctions from excessive current.

## Solid-State Relays

The main advantage of using a mechanical relay is its simplicity and high current and voltage handling capabilities. However, being mechanical, it does present certain problems. The switch contacts are subject to *wear* and *pitting*, as they continually open and close and dirt may build up, causing poor connections over a period of time. Also, the switching time of a mechanical relay is relatively slow.

A *solid-state relay* has no moving parts or contacts to pit and wear out. It can switch from *ON* to *OFF* in the time it takes to turn on a transistor (microseconds).

One example of a solid-state relay interface is shown in Fig. 9-4. The key component in this circuit is the TIL111 *opto-coupler*. This is a *six*-pin integrated circuit containing an *infrared* LED (IRED) and phototransistor. When the computer outputs a 1 to PA1 of the 8255, the 7404 inverter grounds the IRED cathode, causing it to emit light. This, in turn, saturates the phototransistor, causing it to become a *short circuit* between collector (C) and emitter (E). The piezoelectric buzzer then emits a loud 4.8-kHz tone suitable as a small alarm signal.

One disadvantage to the circuit in Fig. 9-4 is that the "relay contacts," pins 4 and 5 of the opto-coupler, can handle only *DC*. Figure 9-5 illustrates a slight modification to this circuit that allows it to control AC as well as DC.

The four diodes in this circuit are connected in a *bridge* configuration and they force the current to flow through the phototransistor in one direction only (DC). In this example the relay contacts are inserted in series with

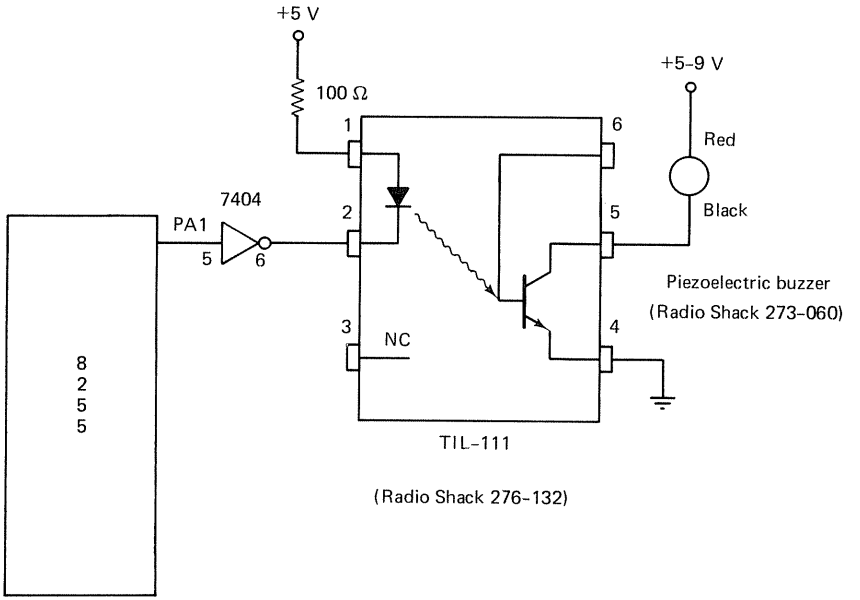



Figure 9-4 A solid-state relay can be built using an *opto-coupler*. When activated by the computer, pins 4 and 5 of the opto-coupler become a short circuit turning on the piezoelectric buzzer. Refer to Fig. 9-3 for the 8255 pin numbers.

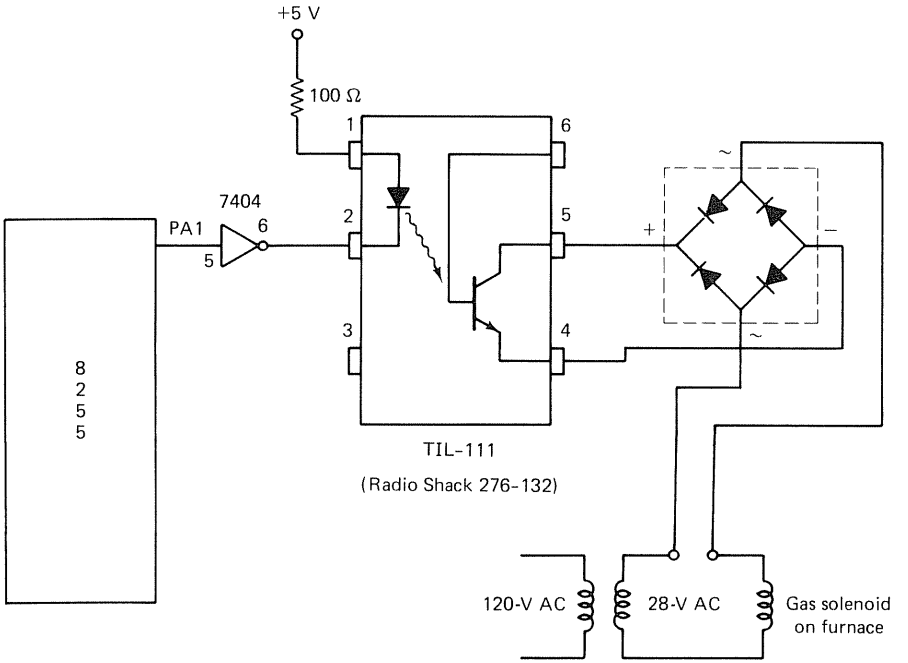



Figure 9-5 Computer-controlled solid-state furnace controller. The four diodes enclosed within the dashed lines are available in a single four-pin bridge rectifier package: for example, Radio Shack 276-1173.

the low-voltage AC control wires to the solenoid on a gas furnace. When the 8255 outputs a 1, the IRED and photoresistor turn *ON*, providing a low-resistance path through the diodes and transistor, turning on the furnace.

Control with BASIC is very simple.

```
10 OUT 236,16        :REM MODEL III ONLY
20 OUT 127,137       :REM INIT PPI. A,B=OUTPUTS, C=INPUT
30 OUT 124,1         :REM TURN ON FURNACE
40 END
```

Commercially available solid-state relays with current capabilities as high as *50 A* are available. They usually contain an opto-isolated *Triac* molded into a plastic package. These are particularly simple to use, as only the input and output terminals are accessible.

### Ultrasonic Interface

An important advantage of using an opto-coupler is the *isolation* that exists between the control and controlled circuits. There is no electrical connection between these two circuits, as only the *light* path is used to pass the ON/OFF control signal. This is important because it protects the control circuit (and the computer!) from faults that may occur in the usually higher voltage controlled circuit.

Another type of circuit that provides even greater isolation than the opto-coupler is the *ultrasonic* interface. The basic idea is illustrated in Fig. 9-6. A *40-kHz* sound wave (ultrasonic) is transmitted, under computer control, through the air. When the receiver detects this signal it turns on a relay, activating the controlled device.

Depending on the transmitter and receiver circuits, distances of 20 to 30 ft between the two are possible. This means that there is no danger of high voltage (120 V AC, for example) somehow inadvertently getting into the computer.

A TRS-80 interface circuit is shown in Fig. 9-7. The transmitter is con-

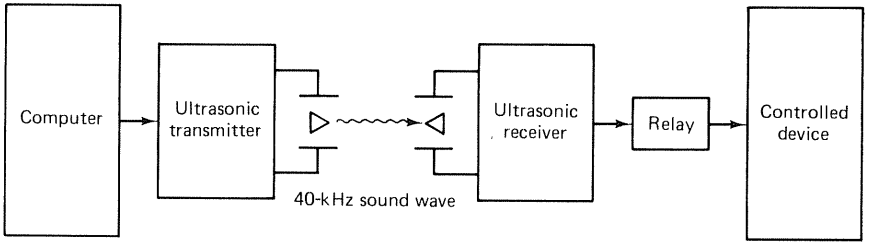
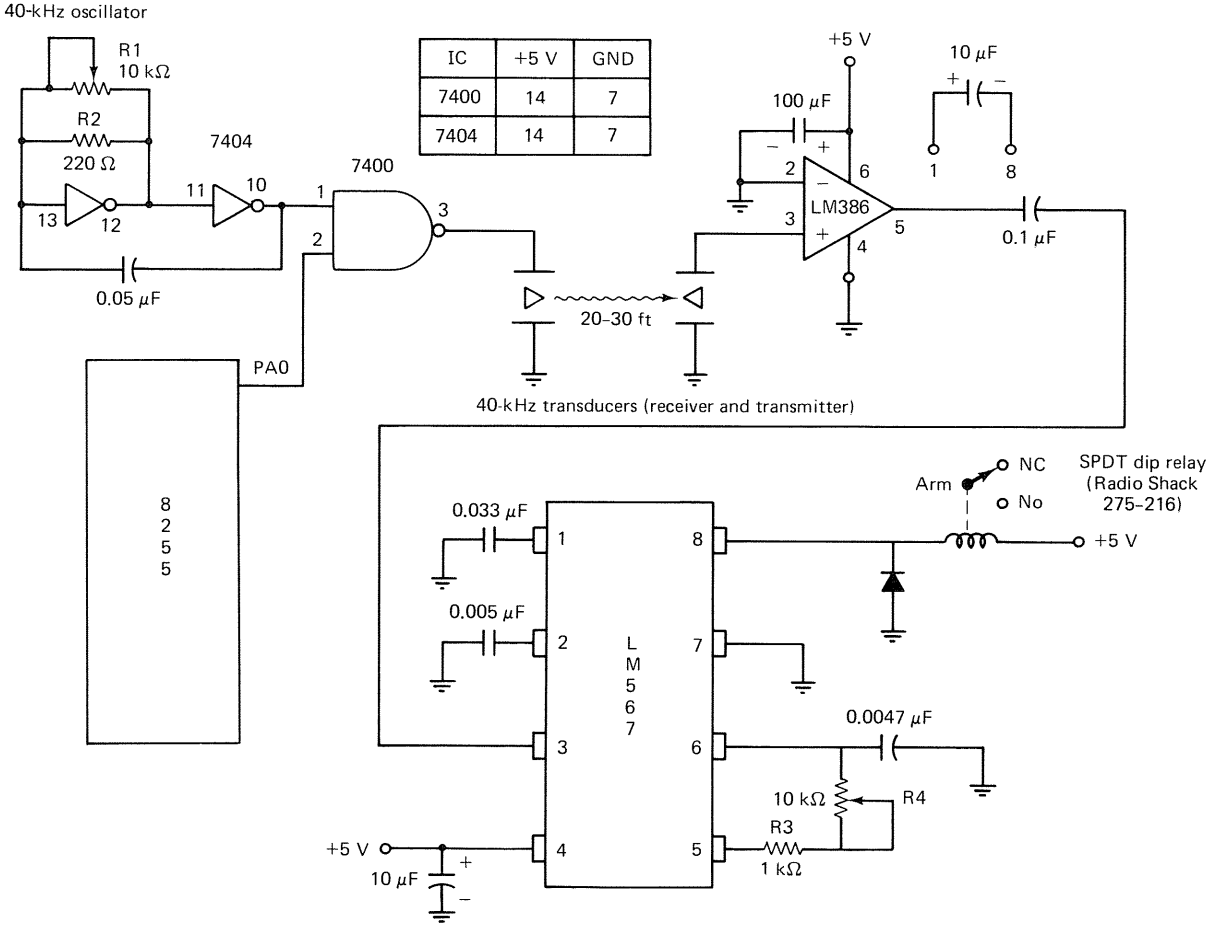



Figure 9-6 Ultrasonic computer interface. The transmitter, when activated by the computer, generates a high-frequency sound wave, which, when received by the receiver, can be used to activate a relay and turn ON or OFF the controlled device.




| IC | +5 V | GND |
|---|---|---|
| 7400 | 14 | 7 |
| 7404 | 14 | 7 |

Figure 9-7 TRS-80 ultrasonic interface. The transmitter and receiver communicate over a 40-kHz ultrasonic sound path. The 40-kHz transducers are available from Micromint Inc., 917 Midway, Woodmere, NY 11598.

trolled by bit PA0 of the 8255. When this bit is a 1, the 40-kHz signal developed by the 7404 oscillator is passed to the output transducer. Because the transducer is optimized for operation at 40 kHz, R1 should be adjusted to produce 40-kHz oscillations at pin 3 of the 7400.

The receiving transducer detects the 40-kHz signal and passes it to the LM386 eight-pin integrated amplifier. Because the received signal is very tiny (a few millivolts at 20 to 30 ft) this amplifier is a *necessity* for distances greater than 3 or 4 ft. The gain of the amplifier is fixed at 20 but may be increased to 200 by adding a 10-$\mu$F capacitor between pins 1 and 8. This may be needed for the longer distances.

The "heart" of the receiver is the LM567 *tone decoder*. When this IC detects the presence of the 40-kHz tone at its input (pin 3), its output at pin 8 switches to ground. What is more, this output pin has the capability of sinking *100 mA* of DC current, making it ideal as a relay driver. In Fig. 9-7 the relay contacts can be connected to a 120-V AC lamp or any other AC or DC appliance. The relay specified is rated for 125 V AC at 1 A, but higher current relays could be substituted.

Although the interface in Fig. 9-7 may appear to provide the *ultimate* in isolation, it does have some limitations. The ultrasonic beam produced by the transmitter is easily *blocked* by a chair, a couch, or your own body (perhaps this suggests its use as a burglar alarm?) and it will not penetrate walls or windows. This means that the transmitter and receiver must generally be in the same room.

Because the circuit described here transmits only an *ON* or an *OFF* message, it is not too practical to have more than one receiver. This is because all receivers would tend to pick up the *same* message.

BSR Ltd. manufactures a complete remote control system that allows placement of up to 16 separate receivers anywhere in your home. This system is similar to the one described here but uses ordinary house wiring to carry the specially coded messages to all receivers. An excellent article describing an interface between this system and the Model I TRS-80 is described by Steve Ciarcia, "Computerize a Home," *BYTE*, January 1980, p. 28.

A similar system offered by Radio Shack interfaces through the Model I or III cassette port. It allows control of up to 256 AC appliances and lights.

## PROCEDURE

**Step 1.** Refer to Fig. 9-3 and wire this circuit on your breadboard. If you still have the hardware from Experiment 8 in place (Fig. 8-3), remove the light and temperature sensors but leave the magnetic switch in place. The 8255 wiring is unchanged. Be sure to wire the relay properly by locating the coil, arm, and normally closed (NC) and normally open (NO) contacts. Connect the moving relay arm to a 6 to 9-V source (the unregulated voltage from

your transformer, for example) or change R2 to 180 Ω (brown-gray-brown) and connect the arm to +5 V.

**Question 9-1.** Write a simple program to test this circuit.

**Step 2.** Connect R2 and the LED to the NC contact and again run the test program from Question 9-1. Note the difference in operation.

**Question 9-2.** With the circuit wired as in step 2, must the relay be *ON* or *OFF* to turn on the LED?

**Step 3.** Refer to Fig. 9-4 and wire this *solid-state* relay and piezoelectric buzzer to the 8255. The 7404 should already be on your breadboard.

**Question 9-3.** The test program used for Question 9-1 will also work for this circuit. To what should the **OUT** command be changed?

**Step 4.** If you still have the magnet switch interfaced to your computer, write a program to accomplish the following:

1. If the magnets are engaged, turn on the relay and LED (reconnect the LED to the NO contact for this step).
2. If the magnets are disengaged, turn off the relay and LED and cause the buzzer to switch *ON* and *OFF* rapidly (an alarm condition).

A solution is provided at the end of this experiment.

*Note.* The following steps involve the ultrasonic interface shown in Fig. 9-7. As explained in the "Discussion" section, optimum performance occurs when the ultrasonic frequency is set to 40 kHz. This requires an *oscilloscope* or *frequency counter* or an accurate *signal generator* that can be substituted for the 7404 oscillator.

**Step 5.** Remove the solid-state and mechanical relay circuits built in the first four steps. Now add a 7400 to your breadboard and wire the ultrasonic transmitter shown in Fig. 9-7. Use two of the extra inverters in the 7404 already on the breadboard.

**Step 6.** For testing purposes enable this circuit with the following program:

```
10 OUT 127,137:   OUT 124,1:   GOTO 10          :REM MODEL I ONLY
```

or

```
10 OUT 236,16:   OUT 127,137:   OUT 124,1:      :REM MODEL III ONLY
   GOTO 10
```

Now, using whatever means are available to you, adjust R1 until the signal at pin 3 of the 7400 is *40 kHz*.

*Note.* The ultrasonic receiver in Fig. 9-7 should ideally be built on a *second* breadboard located *remote* to the transmitter. If this is not possible, wire it on your breadboard together with the transmitter. If you plan to test the circuit beyond distances of 2 to 3 ft, you will have to use the LM386 amplifier. If not, the amplifier can be skipped and the receiving transducer's output connected directly to pin 3 of the LM567.

**Step 7.** With the test program from step 6 holding the transmitter ON, adjust the receiver to 40 kHz by adjusting R4 until the relay clicks ON reliably.

**Step 8.** You may wish to connect an AC light to the relay, some other AC device, or the piezoelectric buzzer and then run the program given in the solution to Question 9-1. Some experimentation with the positioning of the two transducers is needed for best performance.

**Step 9.** Reread the objectives as stated at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

*Note.* The 8255 portion of this circuit will be used again in Experiment 10.

## SOLUTIONS TO QUESTIONS

**9-1.**

```
10 CLS
15 OUT 236,16                          :REM MODEL III ONLY
20 OUT 127,137                         :REM PORTS. A,B=OUTPUTS,
                                        C=INPUT
30 INPUT "DO YOU WANT THE RELAY
   ON OR OFF ";A$
35 OUT 236,16                          :REM MODEL III ONLY
40 IF A$="ON" THEN OUT 124,1 ELSE
   OUT 124,0
50 GOTO 30
```

**9-2.** *OFF.*

**9-3.** OUT 124,2.

STEP 4 (solution)

```
10 CLS
15 OUT 236,16                          :REM MODEL III ONLY
20 OUT 127,137                         :REM INIT PPI
```

```
 30  IF (INP(126) AND 1)=1 THEN OUT 124,1
     ELSE 50
 40  GOTO 30
 50  REM SOUND THE ALARM
 60  OUT 124,2                          :REM BUZZER ON
 70  FOR J=1 TO 50:  NEXT J             :REM WAIT
 80  OUT 124,0                          :REM BUZZER OFF
 90  FOR J=1 TO 50:  NEXT J             :REM WAIT
100  GOTO 30
```

# EXPERIMENT 10

## INTERFACING A DIGITAL-TO-ANALOG CONVERTER

*Note.* To perform this experiment you must have a 0 to 5-V voltmeter to monitor the analog output voltage. Also, the MC1408 converter and LM1458 op-amp require both *negative* (−5 to −15 V) and *positive* (+5 to +15 V) supply voltages.

### OVERVIEW

In this experiment you will wire an *MC1408 digital-to-analog converter* to the TRS-80 expansion bus. The circuit will be calibrated to provide a 0 to 4.98-V programmable power supply in 20-mV steps. BASIC control software will be developed.

### OBJECTIVES

The key points to be learned from this experiment are:

1. A digital-to-analog converter receives a digital word and converts it to a scaled analog voltage, 0 to 5 V, for example.
2. An $n$-bit converter will produce steps of size equal to the full-scale output voltage divided by $2^n$. A typical step size is *20 mV* for an 8-bit converter.
3. A digital-to-analog converter is interfaced to a microcomputer through a normal output port (latches). More than one output port is needed for converters with more than 8 bits.
4. Because BASIC is very slow, *machine language* is required when using the digital-to-analog converter for complex waveform generation (music or speech, for example). BASIC is adequate for applications

such as real-time control of mechanical devices or a programmable power supply.

## PARTS LIST

1 7404 hex inverter
1 7430 8-input NAND gate
1 8255 programmable peripheral interface chip (JAMECO INS 8255)
1 MC1408 (see the text) (JAMECO MC1408L8)
1 LM1458 dual operational amplifier (Radio Shack 276-038)
1 LED
1 270-Ω resistor (red–violet–brown)
2 180-Ω resistors (brown–gray–brown)
2 1-kΩ resistors (brown–black–red)
1 10-kΩ potentiometer
1 100-pF capacitor
1 0 to 5-V voltmeter
1 −5 to −15-V power source

## DISCUSSION

### Digital and Analog Outputs

In Experiments 8 and 9 we discussed digital interfacing techniques. The circuits in these experiments produced discrete *ON/OFF* output levels. Input devices were regarded to switch *ON* or *OFF* when they crossed some preset reference level. In this experiment and the next we do not restrict our inputs and outputs to these discrete levels. Instead, the whole continuum of values between 0 V and some full-scale value will be allowed. If we wish to output 3.73 V or 1.02 V, our circuit will be capable of doing it.

How is this accomplished? In this experiment we introduce a new integrated circuit called the *digital-to-analog converter* (DAC). This IC is capable of accepting an *n*-bit *digital* input word and providing an *analog* output voltage corresponding to the value of this binary word. In Experiment 11 we will use this DAC to form an *analog-to-digital converter* (ADC). With this circuit, an analog input voltage (proportional to temperature, light, pressure, etc.) can be converted to a digital input word and then processed by the computer.

### Digital-to-Analog Techniques

Figure 10-1 illustrates a hypothetical DAC. The circuit receives a binary input which is used to turn on or off a number of weighted voltage sources. A

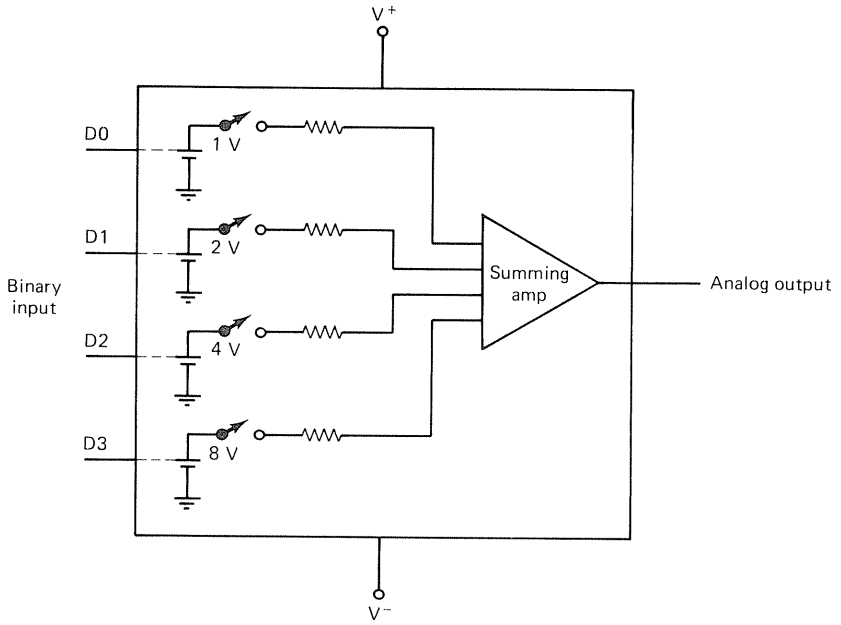



Figure 10-1 Hypothetical 4-bit DAC. The four binary inputs control the four switches. Note that each voltage source is *weighted* according to its bit position. Maximum output occurs when the binary input is 1111 and produces a 15-V (1 V + 2 V + 4 V + 8 V) output.

summing amplifier then adds all voltages to produce the net output voltage. For example, the circuit in Fig. 10-1 will produce a 5-V output when the input word is 0101 (the 4-V and 1-V sources are switched on).

The simple DAC just described has 4 bits and therefore 16 steps from 0 V (0000) to 15 V (1111). The size of one step is always equal to the contribution of the *least significant bit*, 1 V in this case. Knowing the step size, we can predict the output voltage for any binary input as

$$\text{(decimal input value)} \times \text{step size} = \text{analog output voltage} \qquad (10\text{-}1)$$

The *full-scale* output voltage of the circuit in Fig. 10-1 is 16 V even though the output can never reach this value! This is because the most significant bit contributes one-half of full scale, the next bit one-fourth, the next one-eighth, and so on. An *infinite* number of bits is required to actually achieve the full-scale output voltage. For this reason, maximum output is always one step below full scale.

For an $n$-bit DAC the full-scale output voltage is

$$V_{FS} = 2^n \times \text{step size} \qquad (10\text{-}2)$$

**Example 10-1**

For the DAC in Fig. 10-1 determine the full-scale output voltage and output voltage for a 1111 binary input. Assume a 1-V step size.

**Solution** Applying Eq. (10-2), we get

$$V_{FS} = 2^4 \times 1\ \text{V} = 16\ \text{V}$$

and when the binary input is 1111 ($15_{10}$),

$$V_{out} = 15 \times 1\ \text{V} = 15\ \text{V}$$

Often, the full-scale output voltage can be adjusted with external components to whatever value you desire. Then the maximum output voltage will be one step less than this value.

### The MC1408

Figure 10-2 is a data sheet and block diagram of the Motorola *MC1408* 8-bit DAC. This IC is available in several varieties. The MC1408 series will operate from 0 to 75°C, and the *MC1508* will function from −55 to +125°C (the military temperature range). In addition, the MC1408 is subdivided into three *accuracy* levels. The *MC1408L-8* has 8-bit accuracy, the *MC1408L-7* has 7-bit accuracy, and the *MC1408L-6* has only 6-bit accuracy.

Unlike the hypothetical DAC in Fig. 10-1, the MC1408 uses a *weighted current* switching scheme to produce an output *current* that is proportional to the 8-bit binary input. Referring to Fig. 10-2, a 2-mA reference current is established at pin 14 which becomes the *full-scale* output current of the device.

We can rearrange Eq. (10-2) to determine the value of one current step. Substituting $I_{FS}$ (full-scale current) for $V_{FS}$, we obtain

$$\text{step size} = \frac{I_{FS}}{2^n} = \frac{2\ \text{mA}}{256} = 7.8125\ \mu\text{A}$$

This means that maximum output current will be

$$255\ (11111111) \times 7.8125\ \mu\text{A} = 1.992\ \text{mA}$$

The absolute accuracy of these values depends on the part used and the *stability* of the reference current. An MC1408L-6 will be four times *less* accurate than an MC1408L-8.

The transfer characteristics shown in Fig. 10-2 illustrate the output current versus binary input. Note that a binary input of 128 (10000000) will provide an output current of $128 \times 7.8125\ \mu\text{A} = 1\ \text{mA}$ or half of full-scale (128 is half of the 256 possible steps).

### Interfacing the MC1408 to the TRS-80

When interfacing the MC1408 DAC to the TRS-80, three requirements must be met.

## ORDERING INFORMATION

| Device | Temperature Range | Package |
|---|---|---|
| MC1408L6 | 0°C to +75°C | Ceramic DIP |
| MC1408L7 | 0°C to +75°C | Ceramic DIP |
| MC1408L8 | 0°C to +75°C | Ceramic DIP |
| MC1408P6 | 0°C to +75°C | Plastic DIP |
| MC1408P7 | 0°C to +75°C | Plastic DIP |
| MC1408P8 | 0°C to +75°C | Plastic DIP |
| MC1508L8 | −55°C to +125°C | Ceramic DIP |

# MC1408
# MC1508

## Specifications and Applications Information

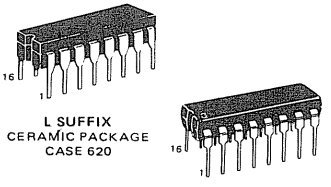
### EIGHT-BIT MULTIPLYING DIGITAL-TO-ANALOG CONVERTER

. . . designed for use where the output current is a linear product of an eight-bit digital word and an analog input voltage.

- Eight-Bit Accuracy Available in Both Temperature Ranges
  Relative Accuracy: ±0.19% Error maximum
  (MC1408L8, MC1408P8, MC1508L8)
- Seven and Six-Bit Accuracy Available with MC1408 Designated by 7 or 6 Suffix after Package Suffix
- Fast Settling Time – 300 ns typical
- Noninverting Digital Inputs are MTTL and CMOS Compatible
- Output Voltage Swing – +0.4 V to -5.0 V
- High-Speed Multiplying Input
  Slew Rate 4.0 mA/μs
- Standard Supply Voltages: +5.0 V and -5.0 V to -15 V

### EIGHT-BIT MULTIPLYING DIGITAL-TO-ANALOG CONVERTER

**SILICON MONOLITHIC INTEGRATED CIRCUIT**




L SUFFIX
CERAMIC PACKAGE
CASE 620

P SUFFIX
PLASTIC PACKAGE
CASE 648

FIGURE 1 – D-to-A TRANSFER CHARACTERISTICS

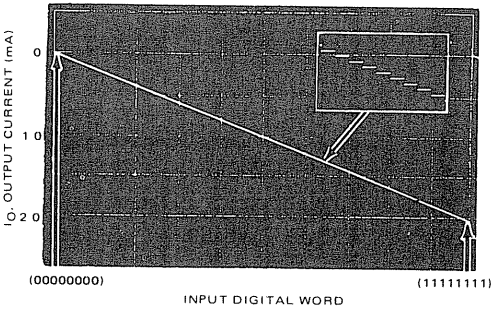
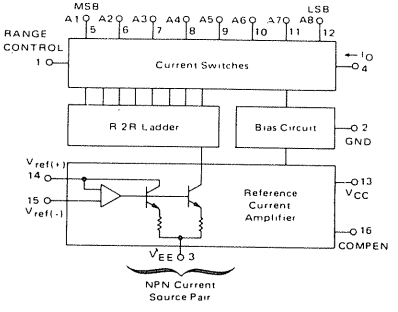



FIGURE 2 – BLOCK DIAGRAM




### TYPICAL APPLICATIONS

- Tracking A-to-D Converters
- Successive Approximation A-to-D Converters
- 2 1/2 Digit Panel Meters and DVM's
- Waveform Synthesis
- Sample and Hold
- Peak Detector
- Programmable Gain and Attenuation
- CRT Character Generation
- Audio Digitizing and Decoding
- Programmable Power Supplies
- Analog-Digital Multiplication
- Digital-Digital Multiplication
- Analog-Digital Division
- Digital Addition and Subtraction
- Speech Compression and Expansion
- Stepping Motor Drive

Figure 10-2 Block diagram and transfer curves for the MC1408 DAC. (Courtesy of Motorola Semiconductor Products Division.)

1. An 8-bit *output port* must be provided. The MC1408 has no internal latches to save the input binary word.
2. A 2-mA *reference current* must be applied to pin 14 of the DAC. This current determines absolute accuracy and stability of the interface.
3. A *current-to-voltage converter* circuit must be used on the DAC output if an output *voltage* instead of current is desired.

A circuit that accomplishes these requirements is shown in Fig. 10-3. Note the following about this circuit:

1. The 8255 is programmed to make port A an output. Data sent to this port will be converted to an analog current by the DAC.
2. The 2-mA reference current is established by the R1R2 voltage divider, which develops 2 V [180 Ω/(180 Ω + 270 Ω) × 5 V] on pin 6 of the LM1458. This op-amp is used as a *voltage follower* passing the 2 V to R3, which establishes the 2 mA.
3. The other half of the LM1458 provides the current-to-voltage conversion. Output current, $I_o$, flows through R4, which can be adjusted to yield any desired full-scale output voltage within the saturation limits of the op-amp.

**Example 10-2**

What value should R4 be adjusted to if full-scale output is to be 5.0 V?

**Solution** Because full-scale output current is 2.0 mA, R4 = 5 V/2 mA = 2.5 kΩ.

**Example 10-3**

What is the *maximum* output voltage for the circuit in Fig. 10-3? Assume that $V_{FS}$ = 5.0 V.

**Solution** Apply Eq. (10-2) to determine the step size:

$$\text{step size} = \frac{V_{FS}}{2^n} = \frac{5.0\text{ V}}{256} = 19.53\text{ mV}$$

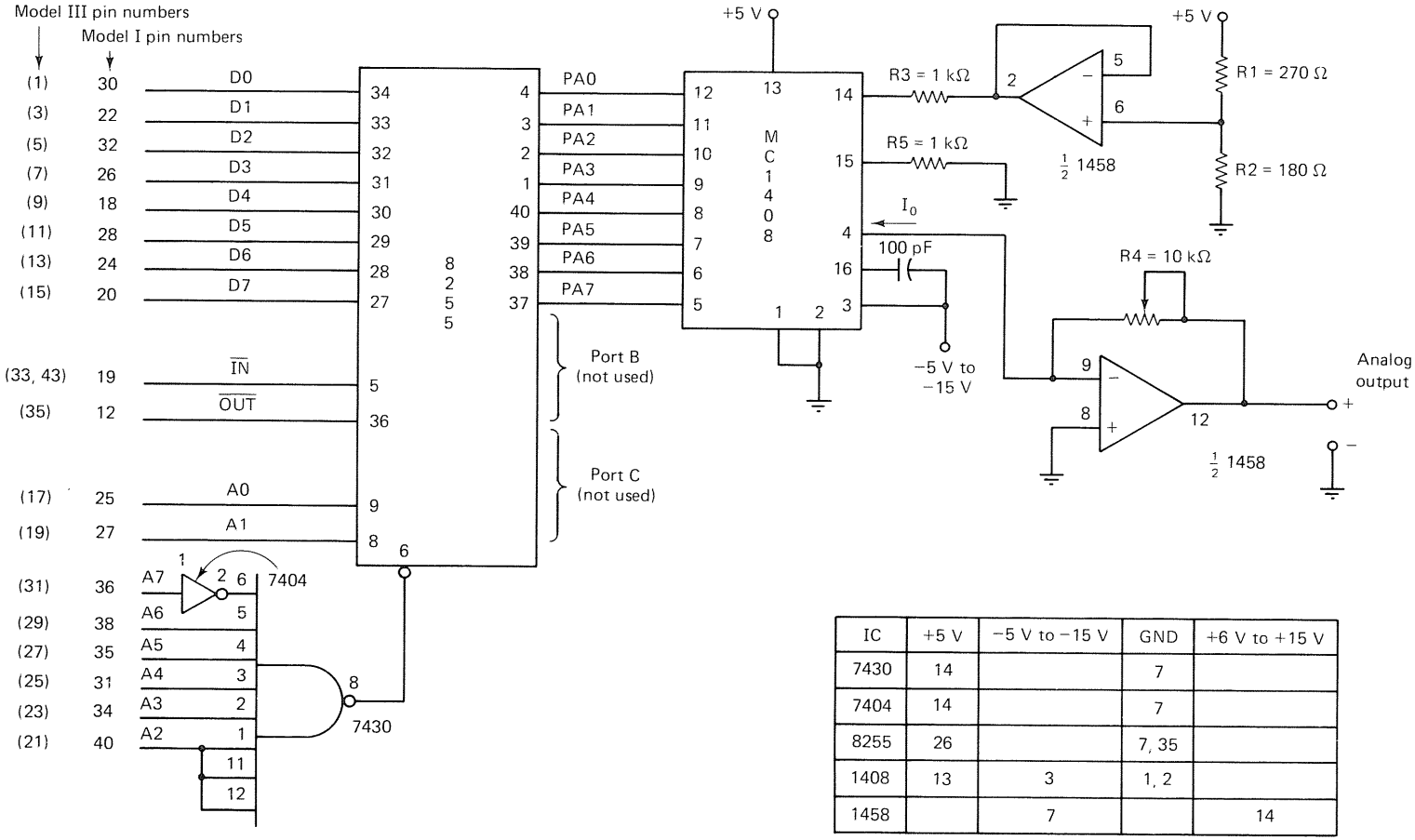
Then when the binary output is 11111111 ($255_{10}$),

$$V_{out}(\text{max}) = 255 \times 19.53\text{ mV} = 4.98\text{ V}$$

The results of Examples 10-2 and 10-3 indicate that, under computer control, we should be able to adjust the output voltage of the circuit in Fig. 10-3 to any voltage between 0 and 4.98 V in approximately 20-mV steps!

## DAC Software

Controlling the DAC in Fig. 10-3 from BASIC is extremely simple. The only requirement is that the desired binary value be output to port A of the 8255.




| IC | +5 V | −5 V to −15 V | GND | +6 V to +15 V |
|---|---|---|---|---|
| 7430 | 14 | | 7 | |
| 7404 | 14 | | 7 | |
| 8255 | 26 | | 7, 35 | |
| 1408 | 13 | 3 | 1, 2 | |
| 1458 | | 7 | | 14 |

Figure 10-3 An 8-bit DAC interface to the TRS-80. Port A of the 8255 is programmed as an output port and the MC1408 converts the binary output word to an analog voltage. The LM1458 op-amp serves as a current-to-voltage converter and 2-mA voltage-follower reference circuit.

For example, to output 2.5 V, the binary code 10000000 ($128_{10}$) should be output.

```
10  OUT 236,16        :REM MODEL III ONLY
20  OUT 127,137       :REM INIT PPI. A,B=OUTPUTS, C=INPUT
30  OUT 124,128       :REM THIS IS PORT A
40  GOTO 40
```

The DAC circuit in Fig. 10-3 must be *calibrated* before it can produce accurate output voltages. This can be accomplished by running the test program just given and adjusting R4 until $V_{out}$ = 2.50 V.

It is also very important that the reference current be *exactly* 2 mA for absolute accuracy. This is established by the voltage-follower circuit connected to pin 14 of the DAC. If the value of either resistor R1 or R2 is incorrect, or the 5-V source is not exactly 5 V, errors will occur. Although the reference in Fig. 10-3 is suitable for our testing purposes, more stable *precision* references should probably be considered for serious applications.

**Example 10-4**

Develop a general formula that can be applied to the circuit in Fig. 10-3 to determine the proper binary code for *any* desired output voltage between 0 and 4.98 V. What code should be used to output 3.65 V?

**Solution** We need only determine what *fraction* of the full-scale output is desired and then scale this result to a number between 0 and 256.

$$\frac{V_{out}}{5.0\text{ V}} = \frac{X}{256}$$

where $X$ is the unknown code. Solving for $X$ yields

$$X = \frac{V_{out}}{5.0\text{ V}} \times 256$$

Now, to produce 3.65 V,

$$X = \frac{3.65\text{ V}}{5.0\text{ V}} \times 256 = 186.88$$

An OUT 124,187 should be used.

One limitation to the circuit in Fig. 10-3 is that the output current is limited to only *20 to 25 mA* by the op-amp. This current capability can be increased fairly simply by the addition of a "pass transistor," as shown in Fig. 10-4. Operation is identical to Fig. 10-3, but the load current will now pass through Q1 instead of the op-amp. Suitable *heat sinking* for Q1 would allow currents in excess of 1 A.

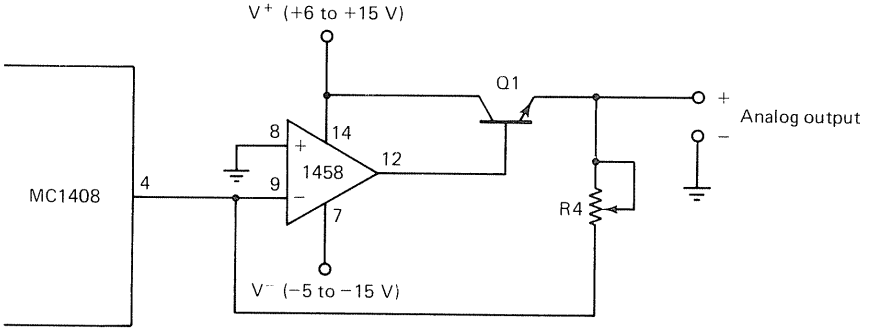



Figure 10-4 This addition to the circuit in Fig. 10-3 increases the current capabilities to several hundred milliamperes or more, depending on the transistor used and heat sinking (if any).

## Other DACs

When designing a digital-to-analog interface circuit, several considerations should be made. Among these are bits of resolution, the need for an external reference circuit, power supply voltages required, internal buffering, and output capabilities (current or voltage outputs). Table 10-1 lists several representative DAC types and their distinguishing characteristics. More information on these devices may be obtained by writing to the manufac-

TABLE 10-1 EXAMPLES OF COMMERCIALLY AVAILABLE 8-, 10-, AND 12-BIT DACS

| Part number | Manufacturer | Power supply | Bits | Internal reference | Internal amplifier | Latched input | Package pins |
|---|---|---|---|---|---|---|---|
| MC1408 | Motorola | +5 V, −15 V | 8 | No | No | No | 16 |
| DAC-08 | Precision Monolithics | ±4.5-±18 V | 8 | No | No | No | 16 |
| MC3410 | Motorola | +5 V, −15 V | 10 | No | No | No | 16 |
| AD7522 | Analog Devices | 5-15 V | 10 | No | No | Yes | 28 |
| AD561 | Analog Devices | 5-15 V, −15 V | 10 | Yes | No | No | 16 |
| DAC-02 | Precision Monolithics | ±15 V | 10 | Yes | Yes | No | 18 |
| AD562 | Analog Devices | 5-15 V, −15 V | 12 | No | No | No | 24 (two-chip hybrid) |
| AD563 | Analog Devices | 5-15 V, −15 V | 12 | Yes | No | No | 24 |

turers indicated in the table. You may also be interested in the *IC Converter Cookbook* by Walter G. Jung (Howard W. Sams & Co., Inc., Indianapolis, IN 46206).

## PROCEDURE

**Step 1.** Refer to Fig. 10-3 and wire the 7430 and 7404 address decoder on your breadboard. Then add the 8255 and connect to the TRS-80 data and control buses. If you are using a Model III computer, be sure to also connect $\overline{\text{IN}}$ (pin 33) to $\overline{\text{EXT I/O SEL}}$ (pin 43) of the socket connector (this portion of the interface may already be in place from Experiment 9).

**Step 2.** Make sure that this portion of the interface works by wiring the simple *logic-level indicator* (logic probe) circuit shown in Fig. 10-5. Touch its input to the port A outputs of the 8255 and run the following program with N = 0 and then 255.

```
10 OUT 236,16      :REM MODEL III ONLY
20 OUT 127,137     :REM INIT PPI. PORTS A,B=OUTPUTS, C=INPUT
30 N=0
40 OUT 124,N       :REM OUTPUT N TO PORT A
50 GOTO 50
```

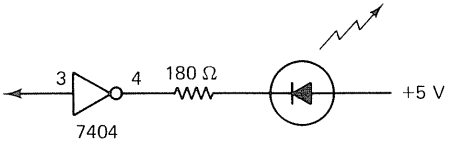



Figure 10-5 Simple logic-level tester using one of the spare inverters in the 7404. A logic 1 on pin 3 will turn *ON* the LED, a 0 will turn it *OFF*.

**Question 10-1.** What should you see with the logic-level indicator for these two values of N?

**Step 3.** When your hardware passes the test in step 2, continue wiring the circuit in Fig. 10-3 by adding the MC1408 and LM1458 op-amp. Connect a *negative* supply voltage (−5 to −15 V) to pin 3 of the MC1408 and pin 7 of the LM1458. Connect +5 V to pin 13 of the MC1408 and your *unregulated* (6 to 15 V) positive supply voltage to pin 14 of the LM1458.

*Note.* By connecting a voltage *greater* than +5 V to the LM1458, its output will be able to swing to at least +5 V. If you use +5 V on pin 14, the op-amp output will be limited to approximately *+4 V* maximum due to saturation of its output stage.

**Step 4.** *Calibrate* the DAC by following the test procedure given under "DAC Software" in the "Discussion" section.

**Question 10-2.** Test the interface by running the calibration program from step 4 with line 30 changed so that you output $24_{10}$, $97_{10}$, $163_{10}$, and $237_{10}$. What output voltage do you measure for each of these values?

**Step 5.** Write a program to make the output voltage *ramp* from 0 to +5 V, dropping back to 0 V, and repeating this cycle. This program should generate a *sawtooth* waveform.

**Question 10-3.** Modify the program in step 5 to produce a *triangular* waveform. The output should ramp from 0 to +5 V and back down to 0 V.

**Question 10-4.** What is the *period* of the triangular waveform generated in Question 10-2? Is it realistic to use a BASIC control program for music generation?

**Step 6.** Develop a BASIC program that will make the DAC perform as a *programmable power supply*. The TRS-80 should prompt you for an output voltage between 0 and 4.98 V and then determine the proper decimal code to output to the DAC to produce that voltage.

*Hint.* Study Example 10-4 for a general formula that can be used to calculate the proper code. Also note that the computer will *truncate* non-integer numbers, so you may want to compensate for this in your program. A solution is provided at the end of this experiment.

**Step 7.** Reread the objectives as stated at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

*Note.* Save your hardware, as it will be used in Experiment 11.

## SOLUTIONS TO QUESTIONS

**10-1.** N = 0; all outputs are low (LED *OFF* at each pin).
N = 255; all outputs are high (LED *ON* at each pin).

**10-2.** 0.47 V, 1.89 V, 3.18 V, 4.63 V.

**10-3.** One possible solution:

```
10  OUT 236,16                  :REM MODEL III ONLY
20  OUT 127,137                 :REM INIT PPI
30  FOR J=0 TO 255              :REM RAMP UP
40  OUT 124,J
50  NEXT J
60  FOR J=254 TO 1 STEP -1      :REM RAMP DOWN
70  OUT 124,J
80  NEXT J
90  GOTO 30
```

10-4. On a Model III the period is approximately 3.5 s (0.3 Hz). Because BASIC takes so long to interpret one program line, it is not practical to generate music with this technique.

STEP 6 (Solution)

```
 10 CLS
 15 OUT 236,16                        :REM MODEL III ONLY
 20 OUT 127,137                       :REM INIT PPI
 30 PRINT@0,
 40 INPUT "ENTER THE DESIRED
    OUTPUT VOLTAGE
    (0-4.98V): ";V
 50 IF V>4.98 OR V<0 THEN 30
 55 V1=(V/5)*256                      :REM SCALE V (0-255)
 60 V2=INT(V1)                        :REM FIND INTEGER VALUE
 70 IF V1-V2 >=.5 THEN V2=V2+1        :REM ROUND OFF TO
                                       NEAREST INTEGER
 80 OUT 236,16                        :REM MODEL III ONLY
 90 OUT 124,V2                        :REM OUTPUT VALUE
100 PRINT@522, "VOUT= ";V;
    " VOLTS"
110 PRINT@550, "DECIMAL
    CODE= ";V2
120 GOTO 30
```

# EXPERIMENT 11

## INTERFACING AN ANALOG-TO-DIGITAL CONVERTER

*Note.* To perform this experiment you must have a 0 to 5-V voltmeter to monitor the analog input voltage. Also, the MC1408 converter and LM1458 op-amp require both *negative* (-5 to -15 V) and *positive* (+5 to +15 V) supply voltages.

### OVERVIEW

In this experiment you will add an LM339 comparator to the hardware constructed in Experiment 10 and program the TRS-80 to become an *analog-to-digital converter*. Using the LM334 temperature sensor from Experiment 8, a digital thermometer will be tested. A BASIC control program to plot temperature on the CRT screen will be given.

### OBJECTIVES

The key points to be learned from this experiment are:

1. An analog-to-digital converter is used to convert an analog input voltage to an *n*-bit digital word.
2. One common conversion technique is to use a digital-to-analog converter in a feedback loop with a digital counter or computer.
3. Two methods for controlling this analog-to-digital converter result in the *successive approximations* and *tracking* converter designs.
4. The step size of an 8-bit converter may be *inadequate* to accurately interface a typical temperature sensor.

## PARTS LIST

See the parts list for Experiment 10 plus the following:

1 220-Ω resistor (red–red–brown)
1 10-kΩ resistor (brown–black–orange)
1 1-kΩ resistor (brown–black–red)
1 LM339 quad comparator (Radio Shack 276-1712)
1 LM334 temperature sensor (Radio Shack 276-1734)
1 10-kΩ potentiometer

## DISCUSSION

### Converting an Analog Voltage to a Digital Input

This experiment will explore the other half of analog interfacing begun in Experiment 10: namely, interfacing an analog input voltage to the TRS-80. Experiment 8 has already shown how non-TTL-level signals may be interfaced, but these voltages were of an *ON/OFF* nature. In this experiment the input voltage will be allowed to have *any* value between 0 V and some full-scale value.

The circuit required to do this conversion is called an *analog-to-digital converter* (ADC). A block diagram representation of a typical converter interface is shown in Fig. 11-1. The ADC receives an input voltage which must be between 0 V and its full-scale value (established by $V_{REF}$) and converts it to an $n$-bit digital word (8 bits in this example). An input port enable pulse is then used to enable the tri-state gates of an input port and gate the data into the computer.

Much of the terminology associated with the digital-to-analog converter (DAC) also relates to the ADC. For example, an $n$-bit ADC will recognize only discrete *steps* of the input voltage. If $n = 4$, there will be 16 unique digital words that can be input from the ADC representing 16 possible values of the input voltage.

**Example 11-1**

Refer to the ADC interface in Fig. 11-1 and determine the step size and digital output for a 1.25-V input. Assume that full-scale voltage is 5.0 V.

**Solution** The concept of step size is identical to that encountered with the DAC in Experiment 10. Using Eq. (10-2), we have

$$\text{step size} = \frac{V_{FS}}{2^n} = \frac{5.0\text{ V}}{256} = 19.53\text{ mV}$$

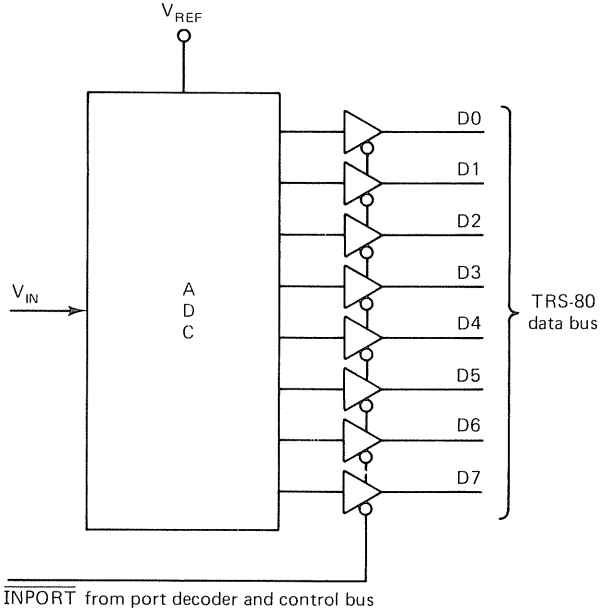



Figure 11-1 Simplified block diagram of an ADC interface. In reality, most ADCs must be "told to make a conversion" before the digital equivalent of the analog input voltage is available to the computer.

If the input voltage is 1.25 V, the binary output will be 1.25 V/5.0 V $\times$ 256 = $64_{10}$ = 01000000.

As this example points out, the ADC is basically treated as a DAC in reverse. You can also see that the full-scale voltage is very critical in determining the binary input word. For some ADCs this value is *preset* and cannot be changed; for others it may be adjustable with an external potentiometer.

One last difference between the DAC and the ADC should be pointed out. The DAC produces a *steady* output voltage as long as its binary input is unchanging. The ADC must generally be told to convert its present analog input to digital. This is referred to as a *conversion*. On some ADC circuits this conversion may take several milliseconds, whereas on others it may take only tens of microseconds. The point is that the ADC does not do this conversion on its own; it must be told to do so by external hardware or software.

### The Flash Converter

Perhaps the simplest way to visualize an analog-to-digital converter is the so-called "flash" converter shown in Fig. 11-2. In this circuit analog comparators compare the input voltage against each possible step value. The output of a comparator is high when $V_{in}$ is greater than the step value and low when less than this value. A logic gate array is needed to convert the comparator outputs to the standard binary code.

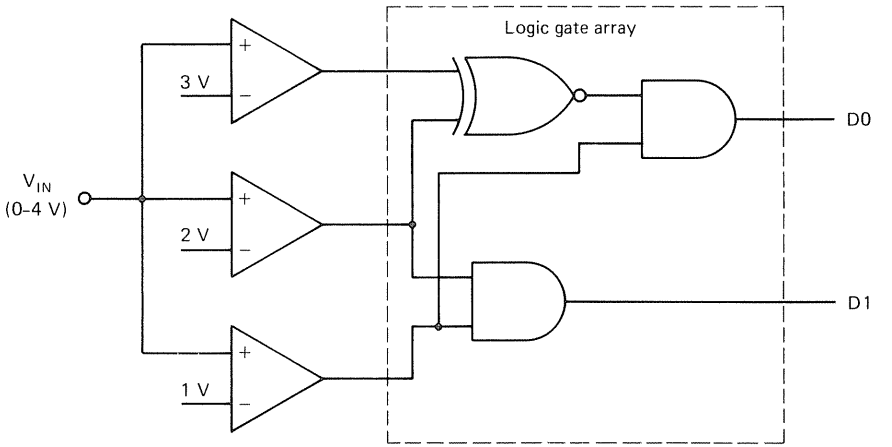



Figure 11-2 A 2-bit "flash" converter. The three comparators detect the three possible steps: 1 V, 2 V, and 3 V. The logic gates convert the comparator outputs to standard binary.

The circuit in Fig. 11-2 is extremely simple because it only provides a 2-bit digital output word, but it does illustrate the concept. A major advantage of this type of ADC is the *conversion speed*, which is limited only by the propagation delays of the comparators and logic gates. This may result in conversion times of less than 1 $\mu$s.

Of course, a 2-bit ADC is not too practical. With 4 V full scale, each step must be 1 V (4 V/$2^2$). If the circuit is expanded to 8 bits, *256* different voltage levels must be detected and *255* comparators will be needed (none is needed for the 0-V step). This is the major disadvantage to this technique. An $n$-bit converter will require $2^n - 1$ comparators. However, when high speed is needed, flash converters may offer the only alternative.

## The Tracking ADC

In this experiment we will use a somewhat more roundabout technique than the flash converter. Because we already know how a DAC works (and have one interfaced to the TRS-80 in Experiment 10), we can use it to output a *known* voltage. This voltage can then be compared against the unknown input voltage and our initial output voltage (or guess) adjusted up or down. Figure 11-3 illustrates this technique.

The computer program makes a first guess at the value of $V_{\text{unknown}}$ by outputting a binary code to the DAC. The DAC converts this to a voltage which is then compared by the analog comparator against $V_{\text{unknown}}$. As Fig. 11-3 illustrates, if the comparator output is high, the program's guess was too big; if low, it was too small. By connecting the comparator output

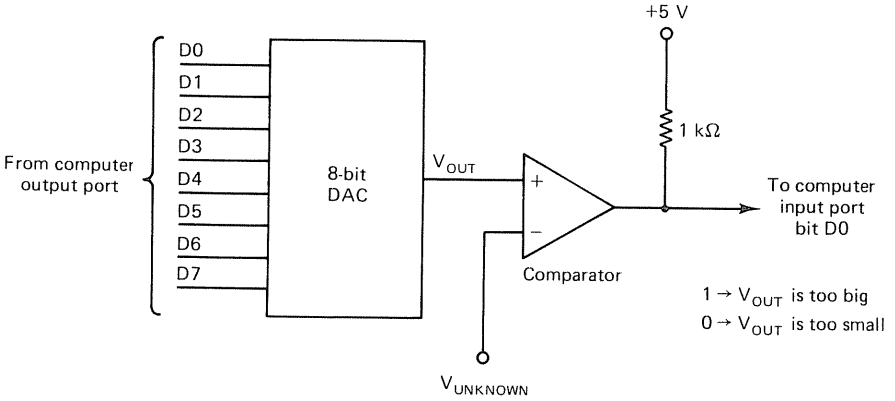



Figure 11-3 An 8-bit ADC suitable for computer control. The unknown input voltage is compared against the DAC's output voltage and the 8-bit output word adjusted up or down until $V_{out}$ and $V_{unknown}$ converge.

to an input port, the control program can tell if the current guess needs to be adjusted up or down.

This type of interface is called a *tracking* analog-to-digital converter and can be built with relatively inexpensive components. It is rather slow and, in this example, ties up the computer to make the conversion process. We can take the computer out of the loop by using the comparator output to control the up/down counting direction of a *binary counter*. This is illustrated in Fig. 11-4.

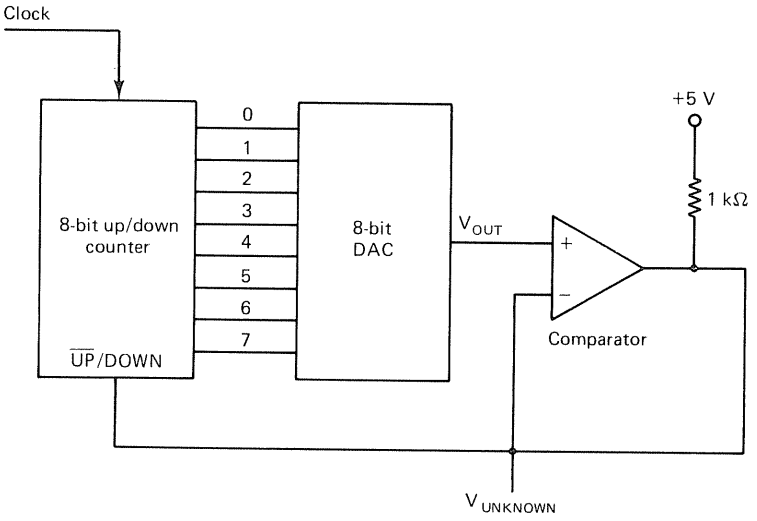



Figure 11-4 Tracking ADC. No computer control is necessary, as the comparator output is used to control the counting direction of an up/down binary counter.

The tracking method of analog-to-digital conversion is not really too "smart." It is analogous to the number-guessing game, in which you must determine a number your opponent is thinking of by asking if a certain number (your guess) is too high or too low. If the range of values is 0 to 255, you might first guess 128 and then 129 and 130 and 131 and so on (assuming that each guess is too low) until finally arriving at the correct number. A *worst-case* scenario has us just finding 255 to be the answer, when the input suddenly switches to 0. It will now take *255* guesses to find this new value! Therefore, in addition to being slow, the tracking ADC has a hard time following analog signals that *change value rapidly* between extremes.

Continuing with the guessing game analogy, we might come up with a better "guessing algorithm." Assuming an 8-bit ADC:

1. As a first guess, turn on the most significant bit, 128 in this case.
2. If this guess is too big, turn off this bit; if not, leave it on.
3. Now turn on the next most significant bit, 64 in this case.
4. Repeat this process until all 8 bits have been tested and one conversion is complete.

The advantage to this technique, called *successive approximations*, is that only $n$ tests are needed for an $n$-bit word compared to $2^n - 1$ tests (worst case) for the tracking method. BASIC control programs for both methods are included in the "Procedure" section. The relative speed of these two techniques will be obvious if you try the experiment.

Don't think a computer must be used for this technique either. Figure 11-5 illustrates the use of an *SAR* (*successive approximation register*) together with a DAC and analog comparator. The SAR can be thought of as a special type of digital counter that follows the sucessive approximations algorithm previously presented.

Using this type of ADC is somewhat different from the DACs presented in the last experiment. Under computer control, the software must monitor the $BUSY/\overline{READY}$ status line of the SAR. When low, the circuit is ready to be read (the conversion is complete). The SAR must also be told when to start a conversion by pulsing the start input. Because most circuits can do this conversion in less than 1 ms, a BASIC control program need not really worry about the $BUSY/\overline{READY}$ status. This is because it will take BASIC several milliseconds just to execute one program line, giving the converter plenty of time to come up with a digital code for the input voltage.

Still other types of ADCs exist. The *dual-slope* (integrating) converter is commonly used in digital *multimeters* and features the advantage of averaging out noise riding on the analog input. For a more complete coverage of

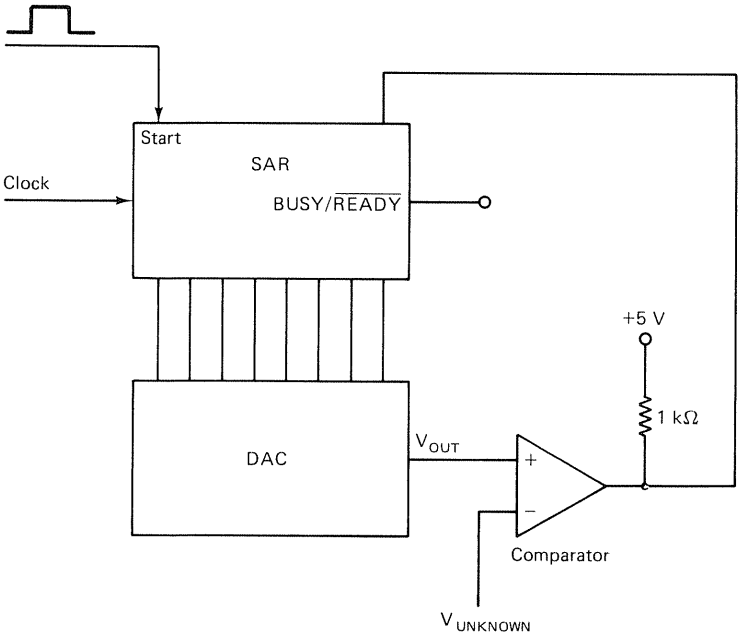



Figure 11-5 Successive approximation ADC circuit. This circuit converges on the proper digital output word much faster than the tracking design.

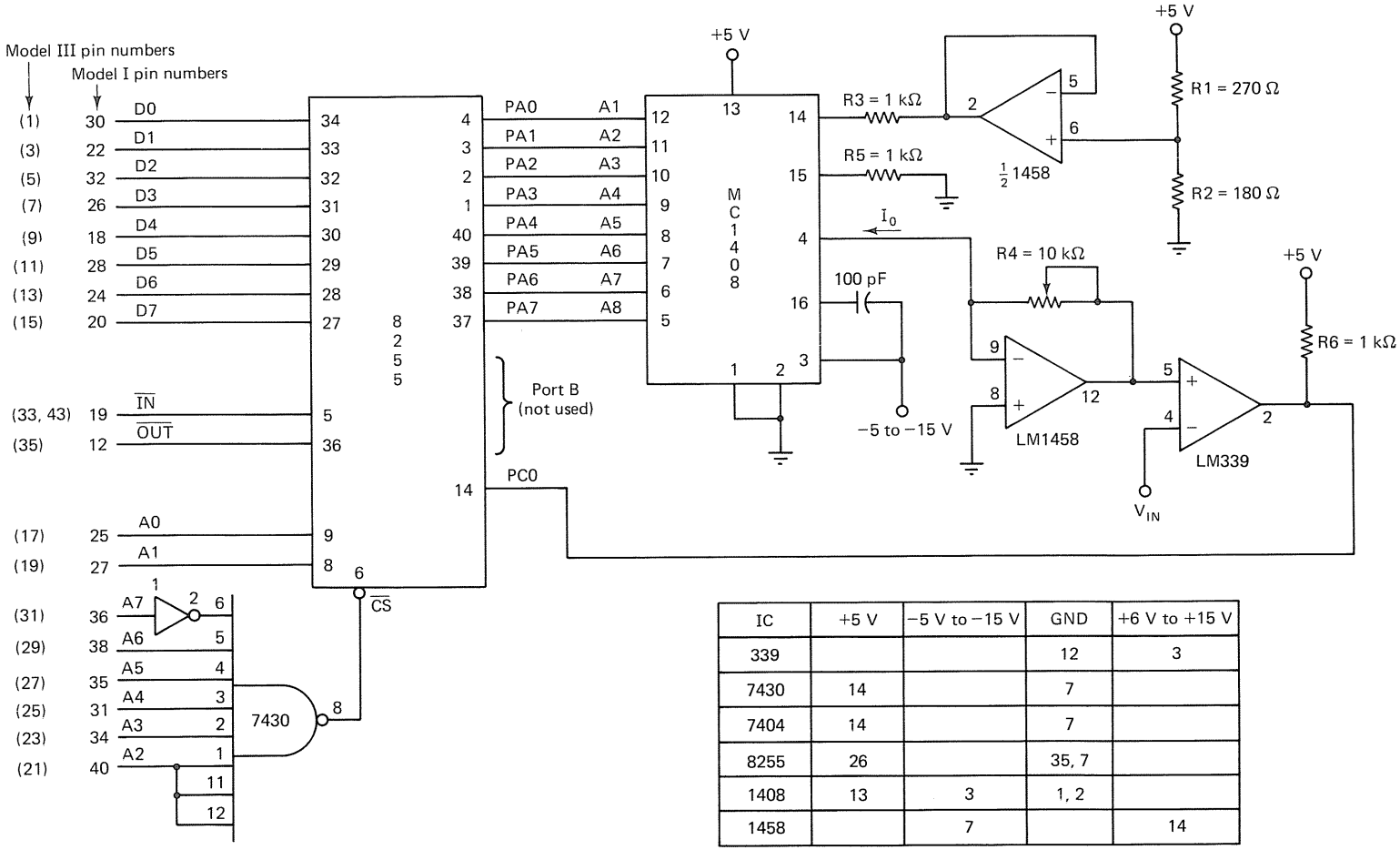
this converter and the others mentioned, refer to the *IC Converter Cookbook* by Walter G. Jung (Howard W. Sams & Co., Inc., Indianapolis, IN 46206).

## PROCEDURE

**Step 1.** Refer to the TRS-80 ADC interface circuit in Fig. 11-6. This is a schematic representation of the block diagram illustrated in Fig. 11-3. You should have all of this circuit wired on your breadboard except for the LM339 comparator. Add this to the interface.

*Note.* This circuit will convert any voltage from 0–4.98 V to digital if the LM1458 op-amp and LM339 comparator positive supply voltages (pins 14 and 3, respectively) are 6 V or greater. If possible, use the *unregulated* voltage (input to the 5-V regulator IC) of your power supply. If you use 5 V for these supplies, your circuit will only have a range of about 0 to 4 V.

**Step 2.** The circuit in Fig. 11-6 is only as accurate as the *reference* voltage applied to the 1-kΩ resistor on pin 14 of the DAC. Measure the voltage on pin 6 of the LM1458. It should be 2.0 V. If not, be sure that R1 and R2 are the proper values.




| IC | +5 V | −5 V to −15 V | GND | +6 V to +15 V |
|---|---|---|---|---|
| 339 | | | 12 | 3 |
| 7430 | 14 | | 7 | |
| 7404 | 14 | | 7 | |
| 8255 | 26 | | 35, 7 | |
| 1408 | 13 | 3 | 1, 2 | |
| 1458 | | 7 | | 14 |

Figure 11-6 TRS-80 analog-to-digital converter interface circuit. This circuit may be used for either the tracking or successive approximations technique.

**Step 3.** As a final calibration step, R4 must be adjusted to provide the proper current-voltage conversion factor. Run the following program and adjust R4 until pin 12 of the LM1458 is at 2.5 V.

```
10 OUT 236,16        :REM MODEL III ONLY
20 OUT 127,137       :REM INIT PPI. A,B=OUTPUTS, C=INPUT
30 OUT 124,128       :REM OUTPUT HALF OF FULL SCALE
40 GOTO 40           :REM NOW ADJUST R4
```

**Step 4.** You may want to be sure the circuit is calibrated properly by running the program from step 6 of Experiment 10 (or the solution provided at the end of that experiment).

*Note.* The comparator function will now be tested. If the DAC is programmed to output 2.5 V, then each time $V_{in}$ exceeds 2.5 V, the LM339 output (and PC0) should go high. The following step programs the TRS-80 to monitor PC0 as $V_{in}$ is varied.

**Step 5.** Connect $V_{in}$ to a variable power source such as the simple potentiometer circuit in Fig. 11-7. Run the following program while monitoring $V_{in}$ with a voltmeter.

```
10 CLS
20 OUT 236,16                          :REM MODEL III ONLY
30 OUT 127,137                         :REM INIT PPI
40 OUT 124,128                         :REM VOUT = 2.5 V
50 C=INP(126) AND 1
60 IF C=1 THEN PRINT@480, "HIGH" ELSE
   PRINT@480, "LOW "
70 GOTO 50
```

**Question 11-1.** When the potentiometer is adjusted so that $V_{in}$ is greater than 2.5 V, what does the program output?

*Note.* We will now program the computer to function as a *digital voltmeter.* Figure 11-8 is a flowchart of the program. The subroutine will use either the *tracking* technique or the *successive* approximation method.

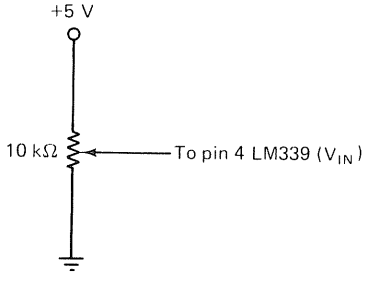



Figure 11-7 Simple 0 to 5-V test voltage for the circuit in Fig. 11-6.

**Step 6.** Refer to the flowcharts in Figs. 11-8 and 11-9a and write a *tracking* ADC program. Run the program and compare it against your voltmeter. A solution is provided at the end of this experiment.

**Question 11-2.** Adjust the input voltage to approximately 5 V and wait for the program output to "stabilize" on the screen. Now adjust the input voltage to 0.5 V. How long does it take for the output to restabilize?

**Question 11-3.** Why does the data flicker so rapidly even after it has *zeroed in* on the input voltage?

**Question 11-4.** In step 6, the TRS-80 is displaying *six* digits of accuracy after the decimal point. Because one step is only about 20 mV (0.02 V) these six digits are actually misleading. How can the program be changed to display only *two* digits after the decimal point?

**Step 7.** Now refer to Fig. 11-9b and write a *successive approximation* subroutine. Insert this in the main program in place of the tracking routine and run it. A solution is provided at the end of this experiment.

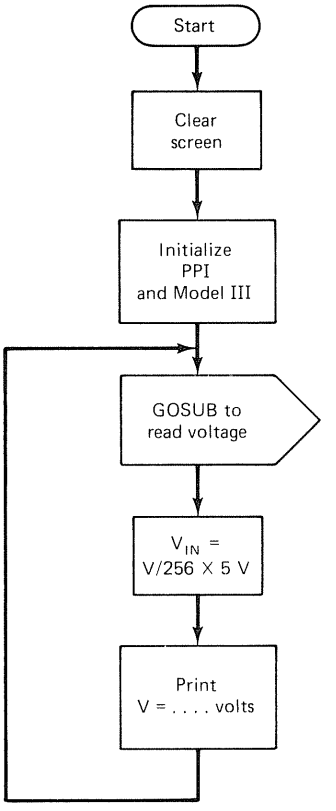



Figure 11-8 Flowchart for the BASIC program that will convert the TRS-80 into a digital voltmeter.

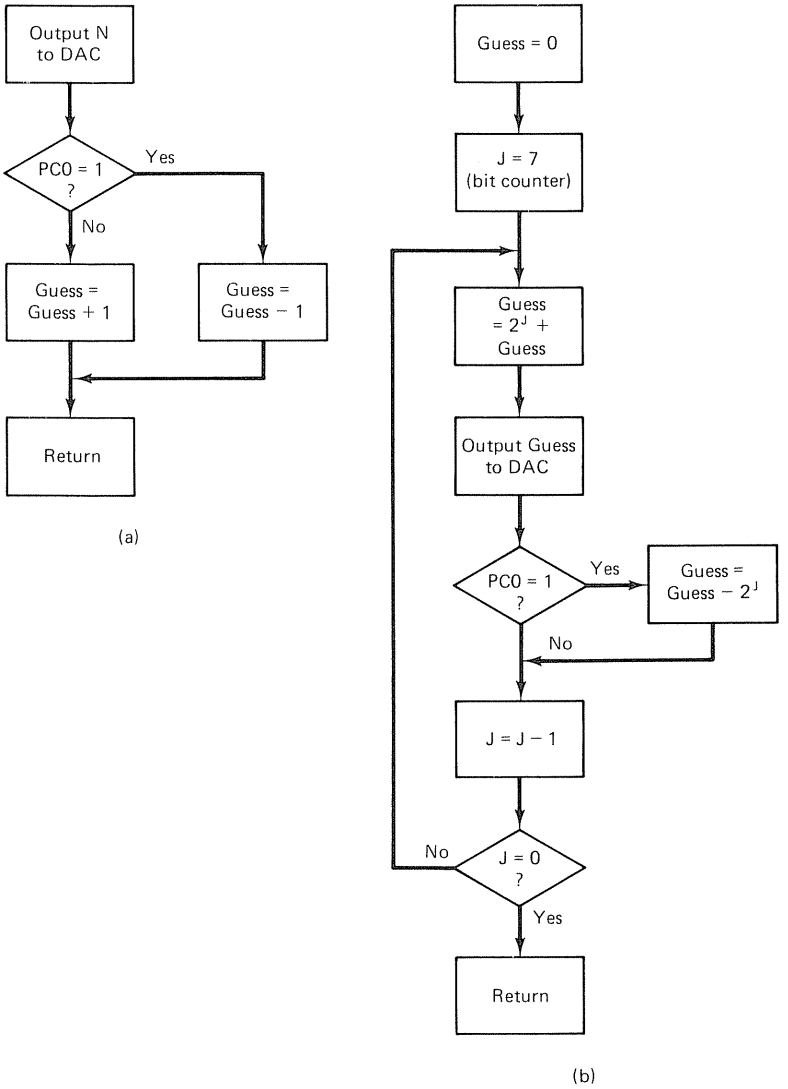



Figure 11-9 Flowcharts for (a) a tracking ADC subroutine and (b) a successive approximation subroutine.

**Question 11-5.** How does the speed of this program compare to the tracking program? Does the conversion time depend on the voltage values?

*Note.* The ADC circuit and software developed could now be used to monitor any number of analog sensors displaying their status on the CRT screen. In these final steps we will use the LM334 *temperature sensor* first used in Experiment 8. The interface cir-

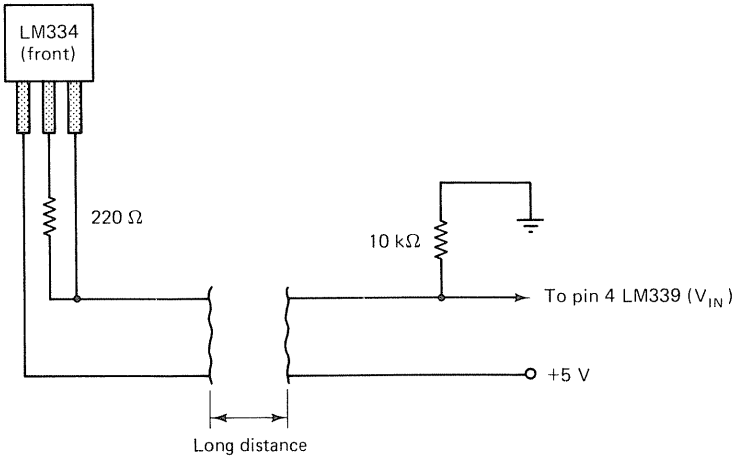



Figure 11-10 The LM334 temperature sensor provides an output voltage that can be converted to degrees Kelvin, Celsius, or Fahrenheit. By connecting the 220-Ω resistor as shown, only two leads are needed for the interface.

cuit is shown in Fig. 11-10. By connecting the 220-Ω resistor as shown, only *two* wires need be run back to the computer. The distance between the computer and the sensor can be most any length without affecting the accuracy (the sensor provides a *constant current* dependent on temperature). The output voltage across the 10-kΩ resistor will vary at the rate 10 mV/°K. In addition, the absolute output voltage will correspond to the sensor's temperature in °K. Therefore, at room temperature (70°F), the output voltage should be 2.94 V (294°K). We can convert this voltage to °F as follows:

$$°F = \{[(V \times 100) - 273] \times \tfrac{9}{5}\} + 32$$

**Step 8.** Use the successive approximation subroutine and modify the ADC program just developed to print the intput voltage and *sensor temperature* in °F. A solution is provided at the end of this experiment.

**Question 11-6.** Watch the voltage display. It will probably *oscillate* between two numbers: for example, 2.99 V and 2.97 V. This corresponds to the 20-mV *step size* of the DAC. Note how the temperature display jumps for these two extremes. For this example (2.99 V and 2.97 V) it would be about 3° from 75 to 78°F. What do you think is the *smallest* temperature change detectable with this circuit?

**Step 9.** (optional) If you have a Model III computer with its built-in *real-time clock*, you may want to try writing a program to plot temperature versus time on the CRT screen. An example of the output of such a program is shown in Fig. 11-11. The time is shown in a 24-hour format. The program was run taking and plotting a temperature sample every half-hour from 10:30 p.m. to 8:30 p.m. the next day. A listing of this program is included at the end of this experiment.

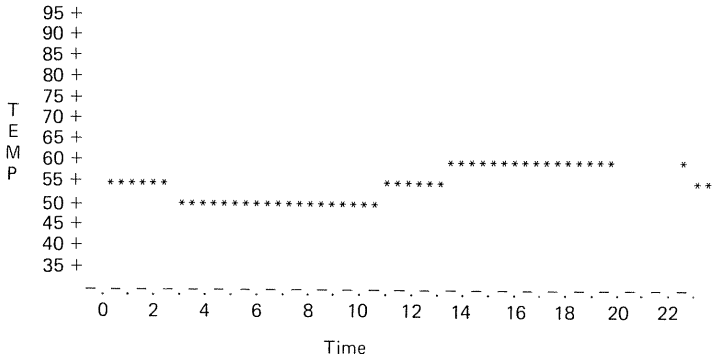



Figure 11-11 Output of the program for step 9 plotting temperature versus time at half-hour intervals.

**Step 10.** Reread the objectives as stated at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

*Note.* The 8255 portion of this interface will be used again in Experiment 12.

## SOLUTIONS TO QUESTIONS

**11-1.** Low. The comparator output is 0 (the DAC output is less than $V_{in}$).

STEP 6 (solution)

```
  5 CLS
 10 OUT 236,16                    :REM MODEL III ONLY
 20 OUT 127,137                   :REM INIT PPI
 30 N=128                         :REM FIRST GUESS IS HALFWAY
 40 GOSUB 700                     :REM TRACKING SUBROUTINE
 50 V=(N/256)*5                   :REM SCALE RESULT TO 0-5 V
 60 PRINT@480, "V= ";V;" VOLTS"
 70 GOTO 40                       :REM DO ANOTHER
                                   CONVERSION
700 REM THIS IS THE TRACKING
    SUBROUTINE
710 OUT 124,N                     :REM OUTPUT THE GUESS
720 REM NOW ADJUST THE
    VALUE OF N
730 IF (INP(126) AND 1)=1 THEN
    N=N-1 ELSE N=N+1
740 RETURN
```

**11-2.** Approximately 13 to 14 s!

**11-3.** With the tracking design the program *never* stabilizes. The input is always one step too high or too low, causing the output to flicker between these two values.

**11-4.** Use the *PRINT USING* command. For example, change line 60 in the solution to step 6 to

```
60 PRINT@480, "V=          VOLTS"
65 PRINT@484, USING "#.##";V;
```

STEP 7 (solution)
Change line 40 in the step 6 solution to

```
GOSUB 800:                          :REM SAR SUBROUTINE
```

and add the following routine:

```
800 REM THIS IS THE SAR
    SUBROUTINE
810 N=0                             :REM N ACCUMULATES THE
                                     ANSWER
820 FOR J=7 TO 0 STEP -1            :REM START WITH BIT D7
830 N=N+2↑J                         :REM ADD TO THE TOTAL
840 OUT 124,N                       :REM SEND TO THE DAC
850 IF (INP(126) AND 1)=1 THEN      :REM IF TOO BIG, DELETE IT
    N=N-2↑J
860 NEXT J                          :REM TEST ALL 8 BITS
870 RETURN
```

**11-5.** The time for one conversion is less than 1 s. Because it does one complete conversion for each call to the subroutine, the actual voltage values do not affect the speed of conversion.

STEP 8 (solution)
Add the following lines to the solution presented for steps 6 and 7.

```
66 T=(((V*100)-273)*1.8)+32
67 PRINT@544, "T=          DEGREES"
68 PRINT@548, USING "### ";T
```

**11-6.** The sensor output changes 10 mV/°K. Because the resolution of the DAC is only 20 mV, a change of 2°K is required before the DAC can detect it. One Kelvin degree corresponds to 1.8 Fahrenheit degrees. Therefore, a change of 2 Kelvin degrees corresponds to a change of 3.6 Fahrenheit degrees, and this is the resolution of the circuit. The accuracy of the LM334 sensor is about ±3%, and this, together with the variation in the DAC reference circuit, combines to make the indicated temperature accurate to only 5 to 10° F.

STEP 9 (solution)

```
   5  CLS:  INPUT "ENTER THE TIME
      AS HOURS, MINUTES ";T1, T2
  10  POKE 16921,T1:  POKE 16920,T2:  :REM SET THE CLOCK
      CLS
  15  OUT 236,16                      :REM MODEL III
  20  OUT 127,137                     :REM INIT PPI
  30  GOSUB 1040                      :REM DRAW VERTICAL AXES
  40  GOSUB 1200                      :REM DRAW HORIZONTAL AXES
  50  GOSUB  800                      :REM 30 MIN ELAPSED?
  55  GOSUB  500                      :REM IF YES, GET THE TEMP
  57  REM PLOT THE DATA POINT
      ON THE SCREEN
  60  X=0                             :REM X IS TIME DISPLACEMENT
  70  FOR  J=35 TO 95 STEP 5
  80  X=X+64
  90  IF T=J THEN J=95:  GOTO 110
 100  IF (T-J)>2 AND T<(J + 5) THEN
      X=X+64:  J=95:  GOTO 110
 105  IF T-J<3 THEN J=95
 110  NEXT J
 120  PRINT@(H-X), "*";
 130  FOR  I=1 TO 100:  NEXT I        :REM MAKE SURE 1S HAS
                                       ELAPSED
 140  GOTO 50
 500  REM **********SUBROUTINE TO
      READ TEMPERATURE*************
 510  N=0
 520  FOR  J=7 TO 0 STEP -1
 530  N=N+2↑J
 540  OUT  124,N
 550  IF (INP(126) AND 1)=1 THEN N=N-2↑J
 560  NEXT J
 570  V=(N/256)*5
 580  T=(((V*100)-273)*1.8)+32
 590  RETURN
 800  REM *************SUBROUTINE
      TO READ TIME*****************
 810  IF PEEK(16919) <> 0 THEN 810
 820  IF PEEK(16920)=0 OR PEEK(16920)=30
      THEN 830 ELSE 810
 830  H=845+PEEK(16921)*2
 840  IF PEEK(16920)=30 THEN H=H+1
 840  RETURN
1040  REM ***********SUBROUTINE TO
      DRAW VERTICAL AXES*********
```

```
1050  X=7
1060  FOR  J=95 TO 35 STEP -5
1070  PRINT@X,J;"+"
1080  X=X+64
1090  NEXT J
1100  X=259
1110  FOR  J=1 TO 4
1120  READ T$
1130  PRINT@X, T$;
1140  X=X+64
1150  NEXT J
1160  RETURN
1200  REM ************SUBROUTINE TO
      DRAW HORIZONTAL AXES******
1210  PRINT@844, "—.—.—.—.—.—.—.—.—.—.—.—.—.—.—.—.—.—.—.—.—.—.—.—."
1220  X=845
1230  FOR  J=0 TO 22 STEP 2
1240  PRINT@(908+(2*J)),J
1250  NEXT J
1260  PRINT@988, "T I M E";
1270  RETURN
2000  DATA "T", "E", "M", "P"
```

# EXPERIMENT 12

## HANDSHAKING I/O

### OVERVIEW

In this experiment you will program the 8255 to operate in *mode 1* (strobed input/output). A DIP switch, 74100 latch, and seven-segment display will be interfaced and used to *simulate* strobed I/O devices. A software seven-segment decoder will be described and tested.

### OBJECTIVES

The key points to be learned from this experiment are:

1. With *unconditional* I/O, the microcomputer may output or input data from its peripherals without regard for their *BUSY/READY* status.
2. *Conditional* I/O requires the microcomputer to monitor the *BUSY/READY* status flag of the I/O device. This is done using either a *polled* or *interrupt*-driven technique.
3. Although polling is simplest to accomplish, it is *slow* and *inefficient* compared to interrupt-driven methods.
4. When operating in mode 1, the 8255 provides several status indicators and strobe inputs that facilitate a *handshaking* I/O interface.

### PARTS LIST

1 8255 programmable peripheral interface (PPI) (JAMECO INS 8255)
1 7404 hex inverter
1 7430 8-input NAND gate

| | |
|---|---|
| 1 | 74100 8-bit latch |
| 1 | seven-segment display DL707 (Radio Shack 276-053) |
| 1 | 8-position DIP switch |
| 1 | red LED |
| 1 | green LED |
| 10 | 1-kΩ resistors (brown-black-red) |
| 2 | 180-Ω resistors (brown-gray-brown) |
| 1 | 68-Ω resistor (blue-gray-black) |
| 2 | pushbutton normally open (PBNO) switches |

## DISCUSSION

### Conditional Versus Unconditional I/O

All the interfacing problems considered to this point have involved *unconditional* I/O in which the input or output device is *assumed ready* to accept or output data unconditionally. This technique is fine for fast peripherals or cases where the operator must be queried from the keyboard before the program continues.

But what about a device like an electronic printer? This peripheral, being electromechanical, requires a finite length of time to print each character it is sent by the microcomputer. An ASR-33 teletype, for example, prints at a rate of one character per *100 ms*. Yet the microcomputer may be capable of outputting data at a rate as fast as one character per *10 μs* (0.01 ms). Obviously, this is much faster than the printer can handle and data will be lost unless some means of *synchronizing* the microcomputer to the much slower mechanical printer is found.

The most common technique is to require the peripheral to supply a *BUSY/READY* status flag that can be read by the microcomputer. In this way, data will still flow between the computer and peripheral, but *conditionally*, the condition being that the status flag must indicate *READY*. Figure 12-1 summarizes these two basic types of I/O.

### Polled I/O

A logical question to ask at this point is: How does the computer know when this *READY* flag is set (that is, when the peripheral is ready)? The most common technique is to establish a *status port* that the microcomputer can periodically *poll*.

Figure 12-2 illustrates the bit assignments for a typical status port. This particular port monitors three *READY* flags, but as many as *eight* could be monitored from a single port. The microcomputer first checks to see (poll)

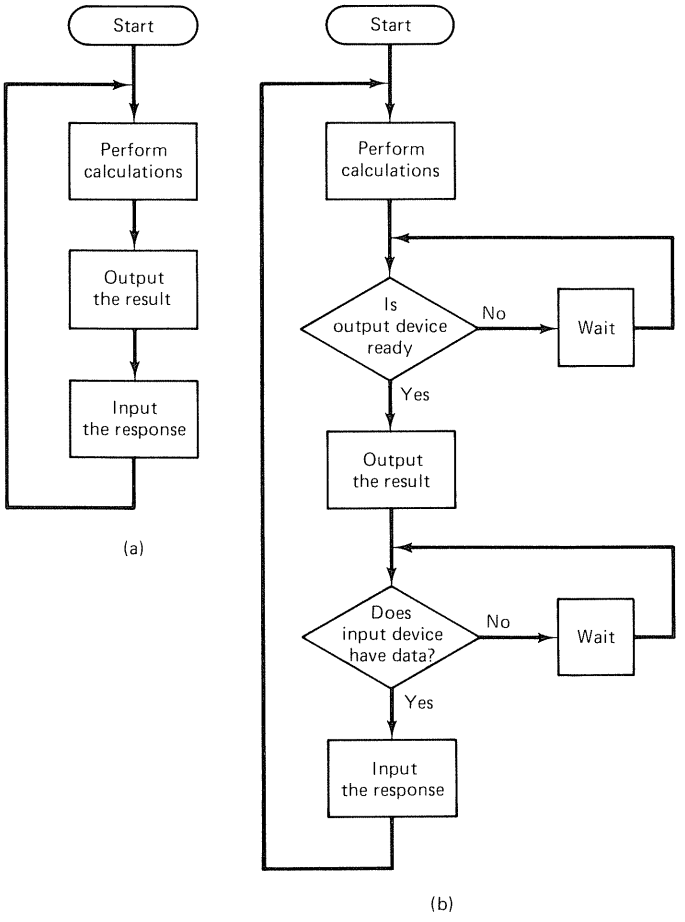



Figure 12-1 Comparing conditional and unconditional I/O. In (a) unconditional I/O is illustrated. The microcomputer assumes that the I/O device is always ready and inputs or outputs data accordingly. In (b) conditional I/O is shown. Now the computer is *synchronized* to the I/O device's *READY* status.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Bit |
|---|---|---|---|---|---|---|---|---|
| | | Keyboard ready | | DAC ready | | | Printer ready | |
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | Value |

Figure 12-2 Bit assignments for a typical status port. One port can provide the *BUSY/READY* status for up to eight different peripherals.

if any of its I/O devices have data or can accept data by testing each bit individually. For example, a digital-to-analog converter (DAC) may take several milliseconds to convert the digital output it receives from the computer to analog. Finally, when ready, its $\overline{BUSY}$/*READY* flag will be set, indicating that it is ready for new data. This condition can be tested by a BASIC program such as:

```
10 IF (INP(124) AND 8) <>8 THEN 10        :REM WAIT FOR READY FLAG
20 OUT 125,Y
30 END
```

Line 10 waits for bit 3 of the status port to go high (the DAC is ready) and then outputs new data to the DAC at data port 125.

A more elaborate polling program could monitor all three flags and branch to different locations when a particular flag becomes ready. For example:

```
10 IF (INP(124) AND 1)=1 THEN 100
20 IF (INP(124) AND 8)=8 THEN 200
30 IF (INP(124) AND 32)=32 THEN 300
40 GOTO 10
```

Special routines to handle each I/O device are assumed to be in locations 100, 200, and 300.

A major *disadvantage* to polling is that the computer is continually tied up checking its I/O devices waiting for a *READY* flag to be set. The analogy has been made that polled I/O is similar to someone waiting for a phone call by continually picking the receiver up and asking if someone is there (rather than letting the bell ring—interrupting you from some other activity). Depending on the circumstances, this may be a rather inefficient use of the computer.

### Interrupt-Driven I/O

The *second* common method of interfacing to conditional I/O devices is to use an *interrupt* technique. As explained in the phone call analogy, it would be more efficient to allow the computer to perform some other job, and only when the I/O device is ready, to stop and service it. This is the essence of interrupt-driven I/O.

Most microprocessors have some form of interrupt capability. You may think of this as a hardwired line directly into the "heart" of the microcomputer. When this line is activated (for example, pulled low), the microprocessor will save the contents of the various registers it is working with in RAM

memory (the *stack*) and branch to a special memory location to service the interrupting device.

Several advantages of interrupt-driven I/O come to mind. First, the microcomputer need only service the I/O device when that device is ready; the rest of the time the computer can be working on some other task. With *vectored* interrupts or *multilevel* interrupts, the memory location of the I/O device service routine is also established at the time of interrupt, so there is no need for a GOTO type of statement to direct the computer to the correct routine. In this way the handling of I/O seems almost "invisible" to the user. In fact, using this technique, the computer can appear to be doing *two* different jobs at one time. For example, a clock circuit could generate an interrupt once each second. The computer could service this interrupt (the *background*) by adding one to a count and display the time of day on the CRT screen. This might require 200 to 300 μs but leave 700,000 to 800,000 μs of time before the *next* interrupt occurs. During this time the computer can work on its main task (the *foreground*). To the user, the computer appears to be doing two tasks simultaneously.

Another advantage of interrupts is that a *priority* structure can be established. If conflicts occur because of *multiple* interrupts, the priority structure will see that the most important device is serviced first. For that matter, the interrupts can be turned off (disabled) altogether when the microcomputer does not want to be interrupted (wouldn't you like to be able to do that!).

The main disadvantage of interrupts is the increased software complexity. The computer can actually become *interrupt bound*, continually servicing interrupts without being able to return to its main task. Debugging errors in this type of software can be extremely difficult.

Generally, the servicing of interrupts must be done with an *assembly language* program, and care must be taken when using this technique with a BASIC-oriented computer such as the TRS-80. User programs may be destroyed and caused to "crash" if the interrupts are not implemented properly.

### Which Scheme Should I Use?

Despite the advantages of interrupt-driven I/O, for most small BASIC-oriented computer systems such as the TRS-80, *polling* may be the best choice. Often in such a system the computer has nothing else to do while waiting for its I/O anyway, and the inefficiency becomes academic. For this type of computer, the interrupt capability is best designed into the overall *operating system*. For example, the real-time clock in the Model III computer is interrupt driven but the software is built into the TRS-80 ROMs.

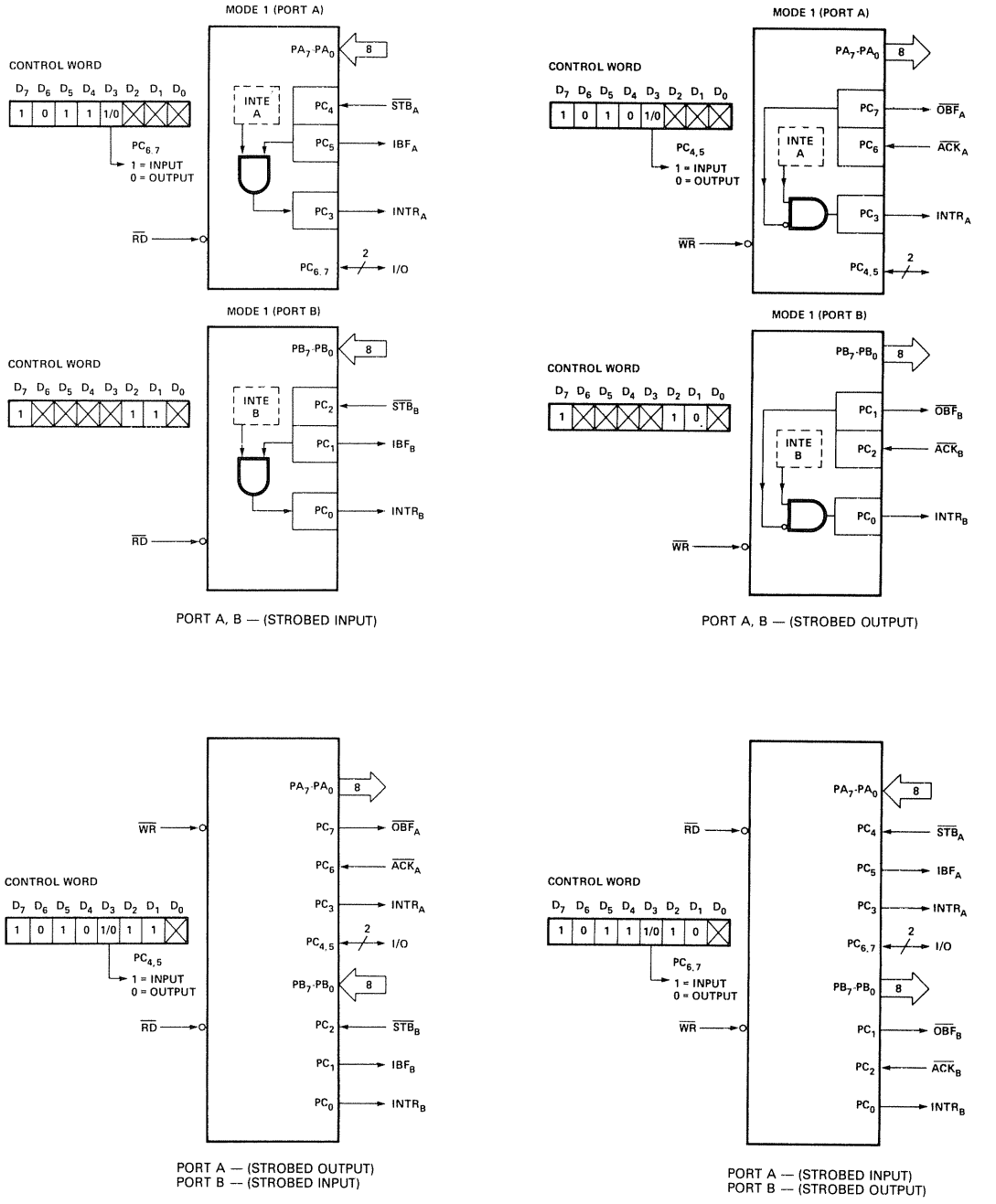



Figure 12-3 The four possible I/O configurations for ports A and B when the 8255 is programmed for mode 1 operation. Port C becomes a *status port* in this mode. (Courtesy of Intel Corporation.)

The 8255 programmable peripheral interface was first introduced in Experiment 5. Since then we have used it in nearly every experiment but always programmed it to operate in *mode 0*. In this experiment we are demonstrating *conditional* I/O and will use the 8255 in *mode 1*. Intel refers to this mode as *strobed input/output*.

Figure 12-3 illustrates the four combinations possible for mode 1 operation. These correspond to the four different combinations of ports A and B as input and output ports. Notice that port C is a *status* port and not a data port for this mode. As such, it provides strobe and acknowledge signals (sometimes referred to as "handshaking" signals) for the two data ports. Figure 12-4 indicates the names and descriptions of the *handshaking* signals.

Input Port

| Signal | Direction | Description |
|---|---|---|
| IBF | OUT | A 1 on this output indicates that the data has been loaded into the input latch; in essence an acknowledgement. IBF is set by the falling edge of the $\overline{STB}$ input and is reset by the rising edge of the $\overline{RD}$ input. |
| $\overline{STB}$ | IN | A 0 on this input loads data into the input latch. |
| INTR | OUT | A 1 on this output can be used to interrupt the CPU when an input device is requesting service. INTR is set by the rising edge of $\overline{STB}$ if IBF is a 1 and INTE is a 1. It is reset by the falling edge of $\overline{RD}$. This procedure allows an input device to request service from the CPU by simply strobing its data into the port. INTE A is controlled by bit set/reset of PC4 and INTE B by bit set/reset of PC2. |

Output Port

| Signal | Direction | Description |
|---|---|---|
| $\overline{OBF}$ | OUT | The $\overline{OBF}$ output will go low to indicate that the CPU has written data out to the specified port. The $\overline{OBF}$ flip-flop will be set by the rising edge of the $\overline{WR}$ input and reset by the falling edge of the $\overline{ACK}$ input signal. |
| $\overline{ACK}$ | IN | A 0 on this input informs the 8255 that the data from port A or port B has been accepted. In essence, a response from the peripheral device indicating that it has received the data output by the CPU. |
| INTR | OUT | A 1 on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set by the rising edge of $\overline{ACK}$ if $\overline{OBF}$ is a 0 and INTE is a 1. It is reset by the falling edge of $\overline{WR}$. INTE A is controlled by bit set/reset of PC6 and INTE B by bit set/reset of PC2. |

**Figure 12-4 Port C of the 8255 supplies several "handshaking" signals when programmed for mode 1 operation.**

Refer to one of the input ports in Fig. 12-3. The input device first pulls the strobe line ($\overline{STB}$) low, which immediately gates data into the 8255 port A or B (but *not* the microcomputer). The 8255 *acknowledges* this data transfer by causing the IBF (*input buffer full*) line to go high. The microcomputer can now use polled or interrupt-driven techniques to determine that data is ready and then input this data. If polling is used, the IBF line of port C (PC1 or PC5) must be tested and data subsequently read from port A or B as appropriate. This read operation also *resets* the IBF flag.

If interrupts are used, the INTR output of the 8255 must be connected to the interrupt input of the microprocessor. Branching to the proper service routine will now occur automatically. Notice that the interrupts must first be *enabled* by a bit set/reset operation in order for INTR to be active.

An output port is handled in a similar fashion. In this case, the microcomputer outputs data to the appropriate 8255 port, causing the $\overline{OBF}$ (*output buffer full*) line to go low. This line is monitored by the output device to determine when data is ready. The output device now *acknowledges* receipt of the data by driving the $\overline{ACK}$ (*acknowledge*) line low.

Again the microcomputer may use polling or interrupts to interface this port. If polling is used, the $\overline{ACK}$ signal of port C (PC2 or PC6) is tested and new data is output only when this line is low, indicating that the *previous* data has been accepted by the output device. The interrupt technique uses the INTR line to determine this same information. Again the INTE line must be high to enable interrupts.

Figure 12-5 will help you determine the proper *control word* to be used to select the port configuration desired. Note that when configuring the 8255, two lines of port C are always available for use as input or output bits independent of the other 6 bits.

**Example 12-1**

Determine the control word to be used to cause port A to be an output, port B an input, and PC4 and PC5 to be outputs.

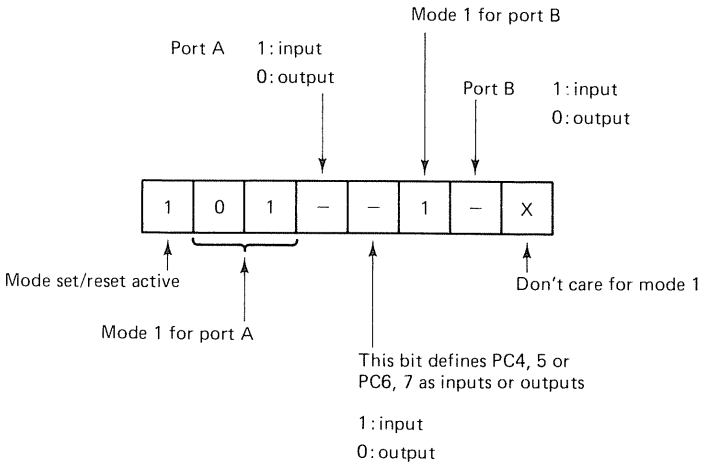
**Solution** Referring to Fig. 12-5, the following binary pattern is found:

$$1010011X = 166 \text{ or } 167_{10}$$

Design Example

In this section we examine a *parallel-printer* interface circuit that requires *handshaking* logic. Printers usually interface to a computer using an RS-232C serial (see Experiment 13) or parallel interface (sometimes referred to as a Centronics interface).

Figure 12-6 illustrates the signal requirements for the parallel (Centronics) interface. First notice that the printer expects to receive *8* bits of data (that is, one parallel word at a time). Data is gated into the printer by a *strobe* pulse, which must go low for 0.5 $\mu$s or longer.




| Port A | Port B | Active bits |
|---|---|---|
| IN | IN | PC6, 7 |
| IN | OUT | PC6, 7 |
| OUT | IN | PC4, 5 |
| OUT | OUT | PC4, 5 |

Figure 12-5 Mode 1 control word format.

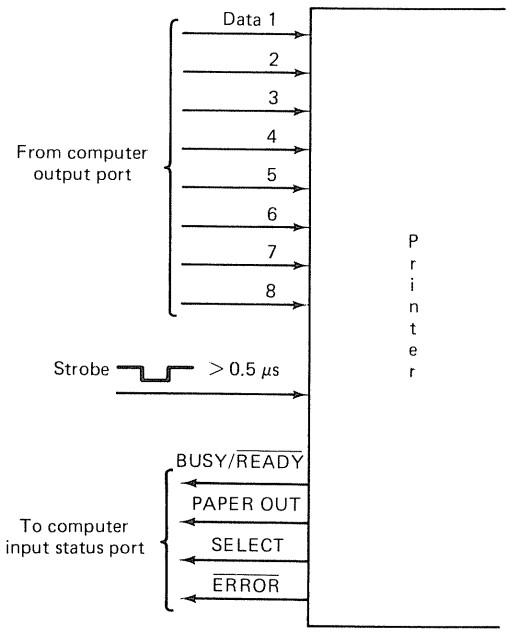



Figure 12-6 Signal requirements for an 8-bit parallel (Centronics) printer interface. Note the handshaking signals required by the printer.

The printer supplies four interfacing signals that comprise the computer input status port but *BUSY/$\overline{READY}$* is the key interfacing signal. Only when this line is *low* should the computer output new data (and a strobe pulse). In this way the computer is *synchronized* to the slow print speed of the printer. The other status signals supplied by the printer should also be monitored by the computer. The *SELECT* (on line/off line) and $\overline{ERROR}$ status signals both must be high and the *PAPER OUT* signal low for normal operation.

Some printers contain *buffers* capable of storing whole blocks of characters. With this technique, the computer outputs bursts of data followed by longer periods of *BUSY* as the printer prints the characters from its buffer.

The TRS-80 has built into its BASIC ROMs routines to communicate with a printer. Two commands may be given from BASIC:

**LLIST or LPRINT**

LLIST causes the current program listing to be sent to the printer port; LPRINT "XXX" directs the computer to output the data between the quotes to the printer port. Because these routines are already stored within your TRS-80, it is a relatively simple matter to add a printer interface to your computer.

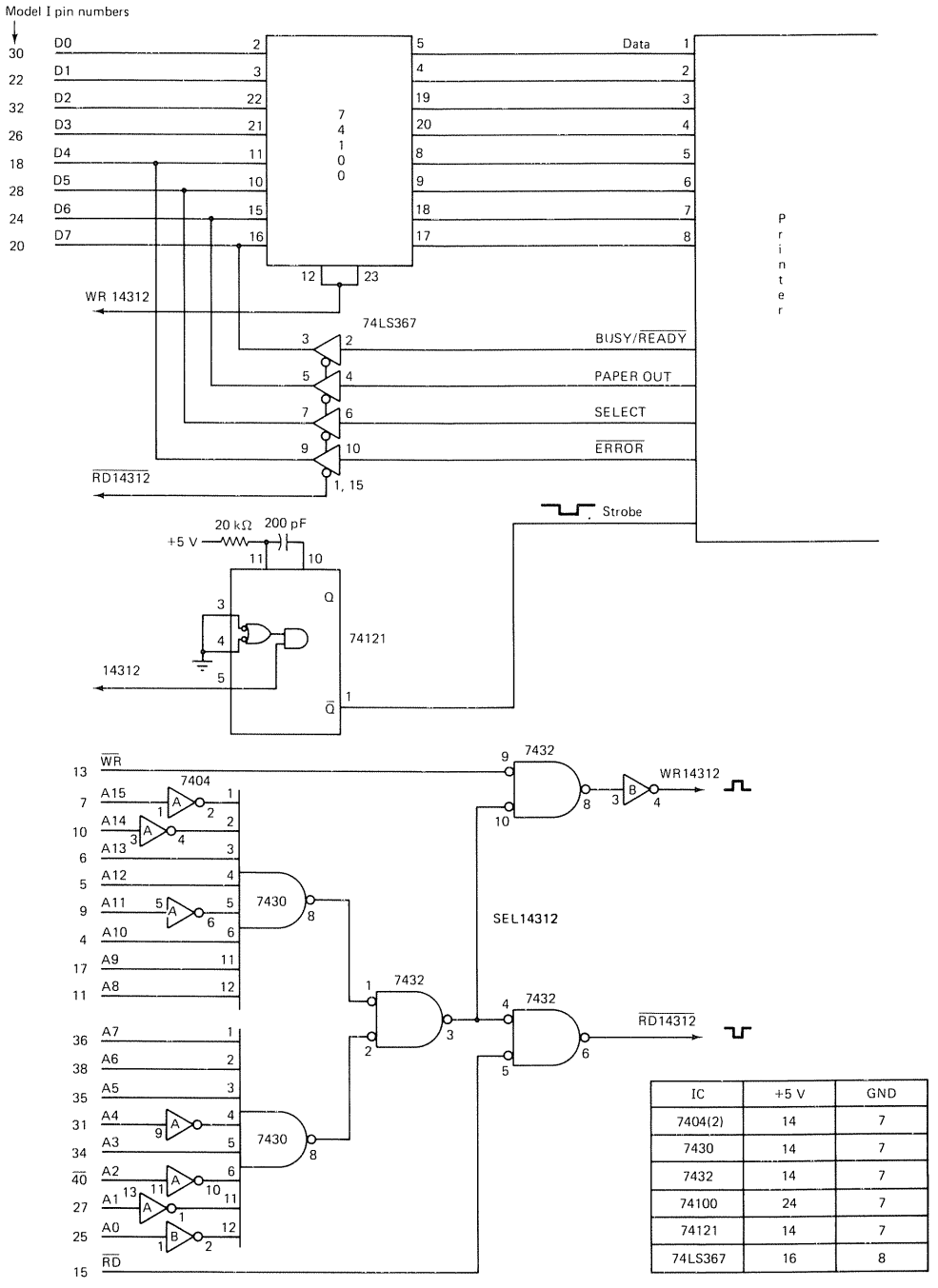
*Note to users of the Model III and Model I with the expansion interface.* Your computer already has the printer interface about to be described. Therefore, there is no need for you to build it. You may still wish to read through this section, as it will help you to a better understanding of handshaking I/O concepts.

The Model I TRS-80 expects the printer interface for the input and output data paths to be *memory mapped* at address $14312_{10}$. Figure 12-7 illustrates a circuit that satisfies the handshaking I/O requirements of the printer and is compatible with the TRS-80 ROM software (that is, LLIST and LPRINT will work properly).

First note that because the interface is *memory mapped*, all *16* address lines must be decoded (you may want to verify that the decoder does indeed recognize address $14312_{10}$). The address select signal (7432, pin 3) is next combined with $\overline{RD}$ and $\overline{WR}$ to generate separate read and write port enable pulses for port 14312. These pulses are then used to enable an 8-bit latch (74100) for the output port and a 4-bit input port (74LS367) for the four status signals.

Because the $\overline{WR14312}$ pulse is *too short* as generated by the TRS-80, a 74121 *one-shot* is used to expand the pulse width to ~2 μs and then used as the printer *strobe* signal.

The TRS-80 ROM printer software will monitor the four status flags and the *BUSY/$\overline{READY}$* flag in particular. When this flag goes low, one character will be output to the 74100 latch and strobed into the printer. Of




| IC | +5 V | GND |
|---|---|---|
| 7404(2) | 14 | 7 |
| 7430 | 14 | 7 |
| 7432 | 14 | 7 |
| 74100 | 24 | 7 |
| 74121 | 14 | 7 |
| 74LS367 | 16 | 8 |

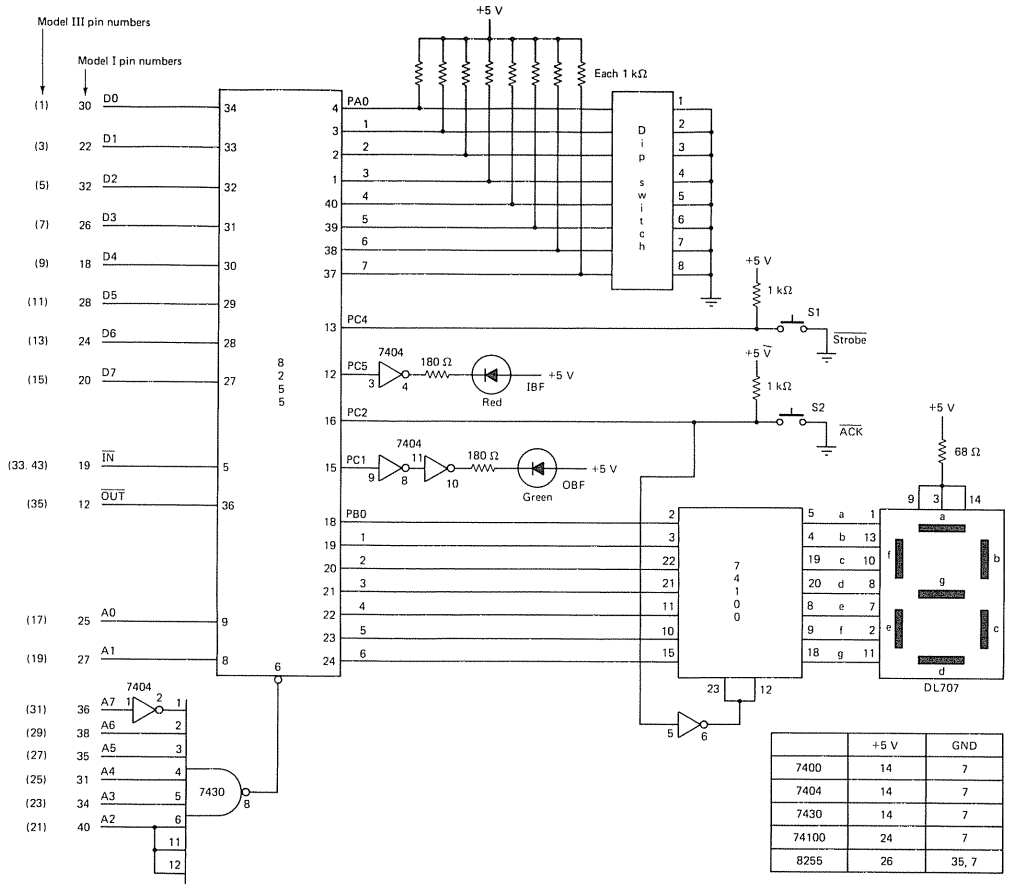
Figure 12-7 TRS-80 parallel (Centronics) printer interface. The circuit includes an 8-bit memory-mapped output port for the printer data and a 4-bit memory-mapped input port for status flags.

course this is all done internally when you type **LLIST** or **LPRINT**, and you need write no software of your own.

## PROCEDURE

**Step 1.** Refer to Fig. 12-8 and wire the 8255 and address decoder on your breadboard (this may already be done from Experiment 11). Now add the port A DIP switch, the **IBF** monitor at PC5, and the strobe input at PC4. The remainder of the circuit will be wired later.

**Question 12-1.** Write a program that configures the PPI in mode 0 and tests the DIP switch connections.




| | +5 V | GND |
|---|---|---|
| 7400 | 14 | 7 |
| 7404 | 14 | 7 |
| 7430 | 14 | 7 |
| 74100 | 24 | 7 |
| 8255 | 26 | 35, 7 |

Figure 12-8 Circuit to illustrate handshaking logic with the 8255.

**Question 12-2.** What control word is needed to configure the PPI for *mode 1* with port A an input port and port B an output port and PC6 and PC7 inputs?

**Step 2.** Load and run the following program. With the program running, set a number up on the DIP switch and depress and release S1. Try this several times for different numbers on the DIP switch.

```
 5 CLS
10 OUT 236,16          :REM MODEL III ONLY
20 OUT 127,188         :REM PPI MODE 1. A=INPUT, B=OUTPUT
30 PRINT INP(124)      :REM READ SWITCH
40 GOTO 30
```

**Question 12-3.** Unless the strobe input is pulsed low, the computer output to the CRT screen remains unchanged. Explain.

**Step 3.** Change the program in step 2 by adding new lines 30 through 60 as follows:

```
30 IF (INP(126) AND 32) <>32 THEN 30      :REM WAIT FOR IBF
40 FOR J=1 TO 500:  NEXT J                :REM TIME DELAY TO SEE
                                           IBF
50 PRINT@460, "THE INPUT VALUE
   IS: ";INP(124)
60 GOTO 30
```

**Question 12-4.** Notice the IBF LED as the program in step 3 is run. What turns *ON* this LED? What turns it *OFF*?

*Note.* The port A interface just constructed provides full handshaking with IBF, the *acknowledge* signal, and the *strobe* input. Also note that the 8255 acts as a *buffer* by holding the input data until the microcomputer is ready to read it.

**Step 4.** Wire the 74100, seven-segment display, $\overline{\text{OBF}}$ LED monitor, and S2 acknowledge switch as shown in Fig. 12-8. Now load and run the following program with S2 depressed. You should observe the numeral 5 on the seven-segment display.

```
 5 CLS
10 OUT 236,16          :REM MODEL III ONLY
20 OUT 127,153         :REM INIT PPI FOR MODE 0. PORT B=OUTPUT
30 OUT 125,18          :REM MAKE SEGMENTS b AND e HIGH (OFF),
                        ALL OTHERS LOW (ON)
40 GOTO 40             :REM HOLD
```

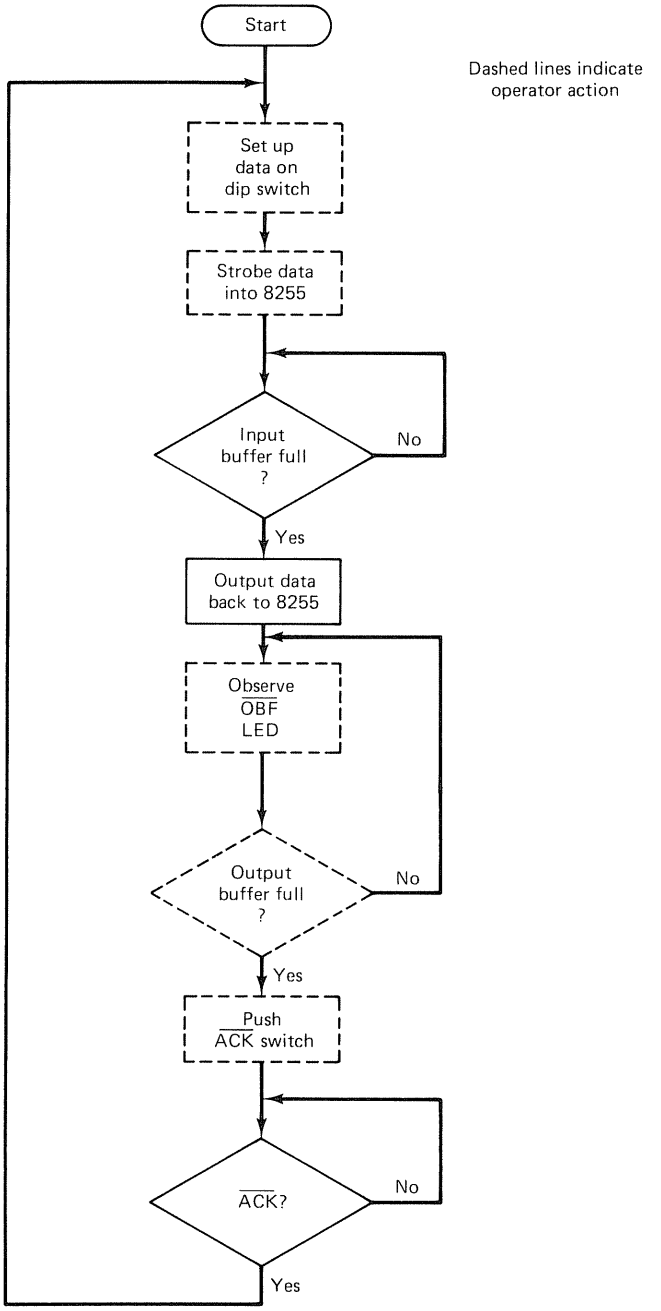



Figure 12-9 Flowchart of the full handshaking program to be written for step 6.

**Question 12-5.** Change the program in step 4 so that the display appears to *count* at a slow rate from 0 through 9. You should then be able to appreciate how *software* can take the place of a conventional hardware seven-segment decoder.

**Step 5.** The following program tests the port B interface using *mode 1*. Load and run this program. Push S2 and observe the $\overline{\text{OBF}}$ LED and display.

```
 5 CLS
10 INPUT "WHAT NUMBER ";N
20 OUT 236,16                    :REM MODEL III ONLY
30 OUT 127,188                   :REM CONFIGURE PPI FOR MODE 1
40 OUT 125,N                     :REM OUTPUT DATA
50 IF (INP(126) AND 4)=4 THEN    :REM WAIT FOR ACK
   50 ELSE 10
```

**Question 12-6.** Notice the $\overline{\text{OBF}}$ LED as the previous program is run. What turns *ON* this LED? What turns it *OFF*?

**Step 6.** Write a program to demonstrate the full handshaking process from inputting of data from the DIP switch to outputting that data back to the display at port B. Figure 12-9 is a flowchart of the process. A solution is provided at the end of this experiment.

**Step 7.** Reread the objectives as stated at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

*Note.* The 8255 portion of this interface will be used again in Experiment 13.

## SOLUTIONS TO QUESTIONS

**12-1.**

```
 5 CLS
10 OUT 236,16          :REM MODEL III ONLY
20 OUT 127,153         :REM CONFIGURE PPI IN MODE 0 WITH
                        PORT A=INPUT, B=OUTPUT
30 PRINT INP(124)
40 GOTO 30
```

*Note.* This program also serves as a binary-to-decimal converter.

**12-2.** 1011110X = 188 or $189_{10}$.

**12-3.** The data is not latched by the 8255 until the strobe input is pulsed low. Until this occurs, port 124 contains the "old" data.

**12-4.** Closing S1 strobes the data into the 8255, turning on the IBF LED. The INP(124) read operation resets the IBF line.

**12-5.**

```
 10 CLS
 20 FOR J=0 TO 9:  READ N(J):          :REM N(J) HOLDS THE
    NEXT J                              SEVEN-SEGMENT
                                        PATTERNS
 30 OUT 236,16                         :REM MODEL III ONLY
 40 OUT 127,153                        :REM INIT PPI
 50 FOR J=0 TO 9
 60 OUT 125,N(J)
 70 FOR I=1 TO 100:  NEXT I            :REM SLOW COUNT
 80 NEXT J
 90 GOTO 50
100 DATA 61,121,36,48,25,18,3,120,0,24 :REM 0-9 PATTERNS
```

**12-6.** An output write to port 125 turns on the $\overline{\text{OBF}}$ LED. It is reset by the acknowledge pulse produced by closing S2. This also clocks the data into the 74100 and seven-segment display.

STEP 6 (solution)

```
 10 CLS
 20 OUT 236,16                         :REM MODEL III ONLY
 30 OUT 127,188                        :REM INIT PPI
 40 IF (INP(126) AND 32) <> 32         :REM WAIT FOR IBF
    THEN 40
 50 Y=INP(124)                         :REM INPUT THE DATA
 60 FOR J=1 TO 500:  NEXT J            :REM TIME DELAY
 70 OUT 125,Y                          :REM NOW OUTPUT THE
                                        DATA
 80 IF (INP(126) AND 4)=4 THEN 80      :REM WAIT FOR ACK
    ELSE 40
```

# EXPERIMENT 13

## SERIAL INTERFACING

### OVERVIEW

In this experiment you will wire a *universal asynchronous receiver-transmitter* (UART) to the I/O bus of the TRS-80. The UART will be wired to transmit to itself and the resulting waveforms will be displayed on the CRT screen.

### OBJECTIVES

The key points to be learned from this experiment are:

1. The UART receives and transmits serial data at a baud rate which is typically *1/16* of the clock rate.
2. The UART is programmed "automatically" to insert start bits, stop bits, and parity bits, and to select the number of data bits per transmitted or received word.
3. A typical UART interface utilizes data "flags" to indicate when a word is ready to be received or transmitted.
4. Most computer terminals use a 7-bit data word and the *ASCII* code for letters, numerals, and punctuation symbols.
5. The UART provides *TTL* input and output levels, but these may be converted to *RS-232C* levels for terminals and printers or converted to two different audio frequencies for transmission over the telephone lines.

## PARTS LIST

1 8255 programmable peripheral interface (JAMECO INS 8255)
1 7430 8-input NAND gate
1 7404 hex inverter
1 LM555 universal timer (Radio Shack 276-1723)
1 AY-5-1013 or TMS-6011 or AY-3-1015 universal asynchronous receiver-transmitter
1 LED
1 1-kΩ resistor (brown–black–red)
1 180-Ω resistor (brown–gray–brown)
1 50-kΩ potentiometer
1 0.01-μF capacitor
1 100-μF capacitor

*Note.* Throughout this experiment the UART referenced will be the General Instruments *AY-5-1013*. Several other UARTS are pin-for-pin compatible, as indicated in the parts list. The *AY-3-1015* requires only a +5-V supply and should be selected if you do not have a −12-V source. Refer to Appendix B for sources.

## DISCUSSION

### Serial Concepts

The microcomputer is inherently a *parallel* machine. It communicates with its memory and peripherals via eight parallel data bus lines. For this reason, all the interfaces we have constructed to this point have been parallel and dealt with these eight data lines.

*Serial interfacing* involves a reorganizing of the data path to a *single* line through which the 8 data bits must pass in a single file. Figure 13-1 compares typical serial and parallel computer ports (in this case to a printer).

Both ports are similar in that normal *address decoding* and *control logic* are needed to place the port at a particular memory or I/O address. In this respect, the microcomputer cannot tell the difference between the two ports. However, what the two ports do with their data once it is received is quite different.

As shown in Fig. 13-1, the parallel port passes the eight lines on to the printer and includes two control lines (*BUSY*/$\overline{READY}$ and *STROBE*) and a common ground, for a total of 11 connections. (In Experiment 12 we saw that the full handshaking interface required three additional lines: $\overline{ERROR}$, *SELECT*, and *PAPER OUT*.)

The serial port, on the other hand, converts the data to single bits

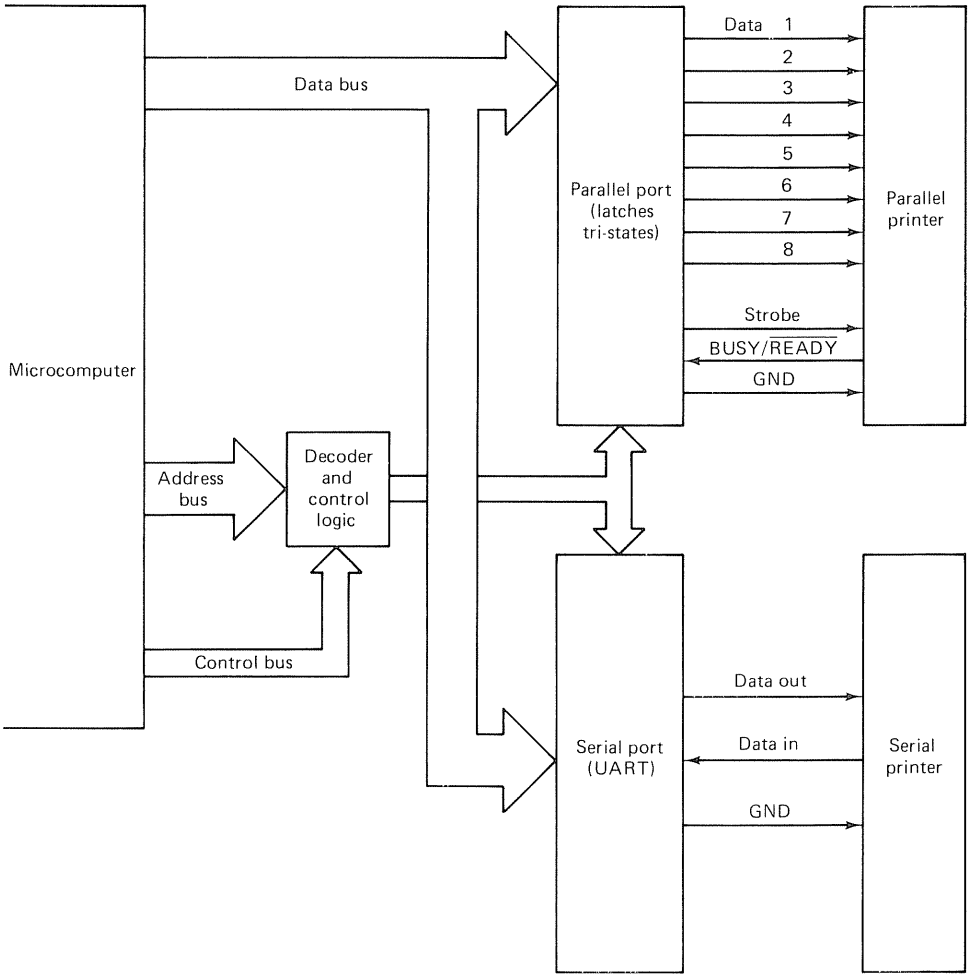



Figure 13-1 Comparing serial and parallel computer ports. The parallel port is much faster but also more complex than the serial port.

which are transmitted one at a time to the printer using the *DATA OUT* line. If the printer wants to reply, it may do so in a similar fashion using the *DATA IN* line. Finally, a common ground establishes the third connection.

The advantage of the serial technique is its *simplicity* and *low cost*—only a three-conductor cable is needed. The advantage of the parallel technique is *high speed*, as all data bits arrive at once.

The choice of which method to use usually depends on the *distance* between the computer and peripheral (lengthy multiconductor cables can be very expensive) and the *speed* of transmission required. In many, if not most, the peripheral is a human interface (printer, CRT terminal, etc.) and

extreme high speed is not critical. In these cases the economy of serial communications is chosen.

Another advantage to the serial technique is that it may be used with a *modem* (modulator-demodulator) to pass data over the telephone lines for *thousands* of miles. There is more detail on this technique in the last section of the "Discussion."

### Baud Rate

The rate at which the serial data bits are transmitted is referred to as the *baud* rate. A typical baud rate for a computer terminal is 1200, meaning that 1200 bits are transmitted each second. Although you might think that this would represent 150 *bytes* per second (1200/8), in practice, *extra* bits are sent along with the data bits, so that one character (or byte) may actually be as many as *12* bits long. This reduces the *character rate* to 100 characters per second in this example, but the baud rate remains at 1200.

Several factors affect the choice of baud rate for a given interface. Because a printer is a mechanical device, we must be careful not to send it characters faster than it can type them out, or data will be lost. For example, the ASR-33 teletype, which runs at 110 baud, is a very well known reliable terminal. But it can accept data at a rate of only *10* characters per second.

When long cables are used (>1000 ft) the internal *capacitance* of the cable limits the maximum speed. Baud rates as high as *19,200* can be used with CRT terminals and short (<30 ft) cables.

### The UART

When comparing the serial and parallel ports in Fig. 13-1, it was stated that the serial port is less complex and more economical than the parallel port. However, you might question this since the serial port must convert the parallel data from the computer to a serial format. Fortunately, the IC manufacturers have recognized the importance of a circuit capable of doing this conversion, and have developed a special serial communications chip called the *universal asynchronous receiver-transmitter*, or UART for short.

The UART contains a separate and independent *transmitter* and *receiver* for serial data in one IC package. All that is needed is to add a clock circuit which establishes the baud rate. Usually, this clock runs at *16* times the actual baud rate (for example, at 1200 baud the actual clock rate is $16 \times 1200 = 199.2$ kHz). Figure 13-2 illustrates the key components in a typical UART.

The transmitter section receives 8 bits of data from the computer and begins transmitting the data when the *strobe* pulse occurs. The control section determines how many bits each transmitted word will have, the number

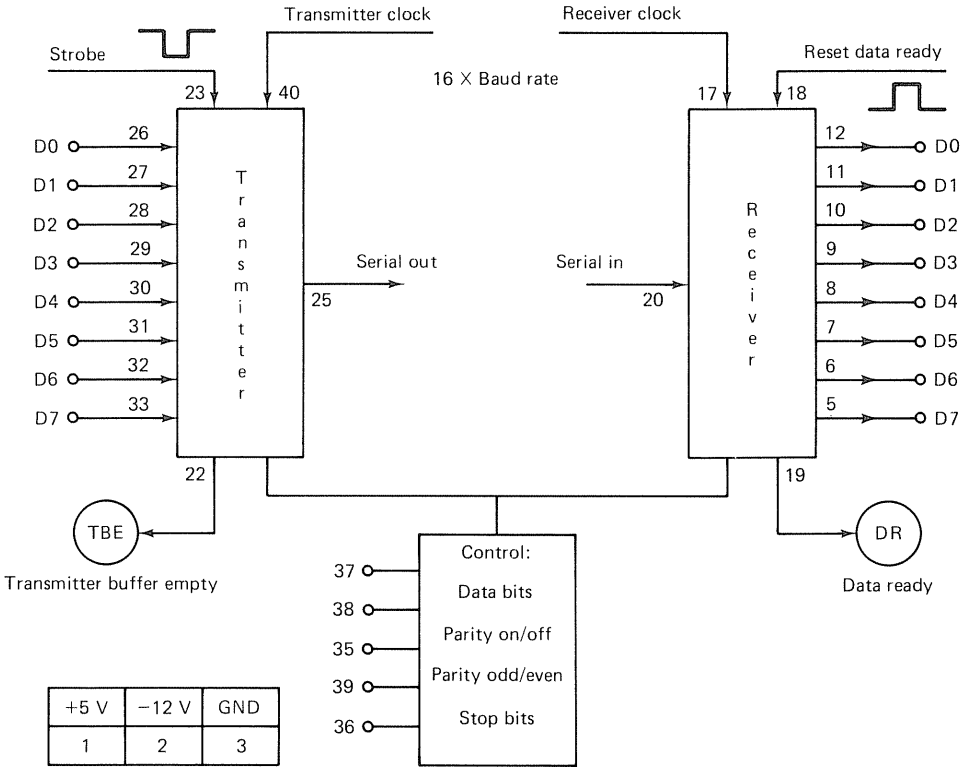



Figure 13-2 Block diagram of a UART. A separate receiver and transmitter are provided. Hardwiring control pins high or low determines the number of data bits per word, stop bits, and parity. The pin numbers shown are for the AY-5-1013 UART used in this experiment.

of stop bits, and the choice of parity. The *transmitter buffer empty* flag (TBE) indicates if the transmitter is ready for a new data word.

Similarly, the receiver *data ready* flag (DR) indicates that a serial character has been received and is available to be read. Once the data has been read, the data ready flag should be reset by applying a *reset data ready* pulse as shown in Fig. 13-2. The control section is shared by the receiver and transmitter, so the word configuration must be the same for both.

## Data Word Format

It was mentioned earlier that several extra bits are transmitted together with the data. These are the *start* bit—always a 0, one or two *stop* bits—each always a 1, and an optional *parity* bit. The parity bit may be chosen to be

*odd* or *even*. This simply means that the *total* number of 1's in the word (including the parity bit but excluding any stop bits) will be even or odd.

**Example 13-1**

Draw waveforms to scale for the output of a UART running at 1200 baud and transmitting the byte 01010101. Assume odd parity and 2 stop bits.

**Solution** Refer to Fig. 13-3. Each character begins with a start bit and ends with the stop bit or bits. This maintains synchronization and allows the receiver to locate the start of the word. Next the 8 data bits follow, least significant bit first. The parity bit is next and is a 1 to maintain an *odd* number of 1's in the word (five in this case). Finally, 2 stop bits end the word.

At 1200 baud, one bit lasts for 1/1200 s = 833 $\mu$s. It takes 12/1200 s = 10 ms to transmit the entire word. This corresponds to 100 *characters* per second.

The UART shown in Fig. 13-2 is typical of the *General Instruments AY-5-1013*. This type of UART allows most of the control functions to be *hardwired* high or low and simplifies software control.

Another type of UART, typified by the *Intel 8251* or *Motorola 6850*, is similar to the 8255 programmable peripheral interface chip. These devices are controlled totally by *software*. In either of these two devices the only data path is the bidirectional 8-bit data bus. Received and transmitted data is sent over these lines, as is control information establishing parity, number of stop bits, and data bits per word. The interested reader is referred to the

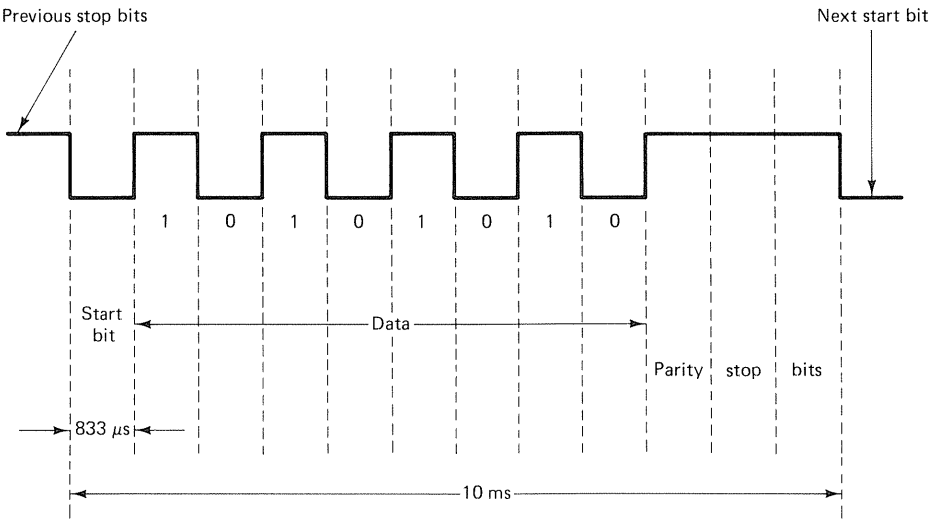



Figure 13-3 Output waveforms for a UART transmitting the byte 01010101 with odd parity and 2 stop bits. The baud rate is 1200.

appropriate data sheets supplied by Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051, and Motorola Semiconductor Products, Inc., P.O. Box 20912, Phoenix, AZ 85036.

### Interfacing to the TRS-80

Although there are many ways in which the AY-5-1013 UART can be interfaced to the TRS-80 I/O bus, perhaps one of the most versatile approaches is to use the three *built-in* ports of the 8255. This design is shown in Fig. 13-4. Port A receives data from the UART when the *DATA READY* flag (**DR**) connected to PC2 goes high. Similarly, when the *TRANSMITTER BUFFER EMPTY* flag (**TBE**) connected to PC1 goes high, we may transmit a new data word to the UART over port B. Port C is used to monitor flags and control UART operation, as detailed in Table 13-1.

Pins 37 and 38 of the UART determine the number of bits per data word, as shown in Fig. 13-4. These pins are *hardwired* for this interface to select a 7-bit data word. There are still several other pins which are either not needed for the interface in Fig. 13-4 or are hardwired high or low. These are explained in Table 13-2.

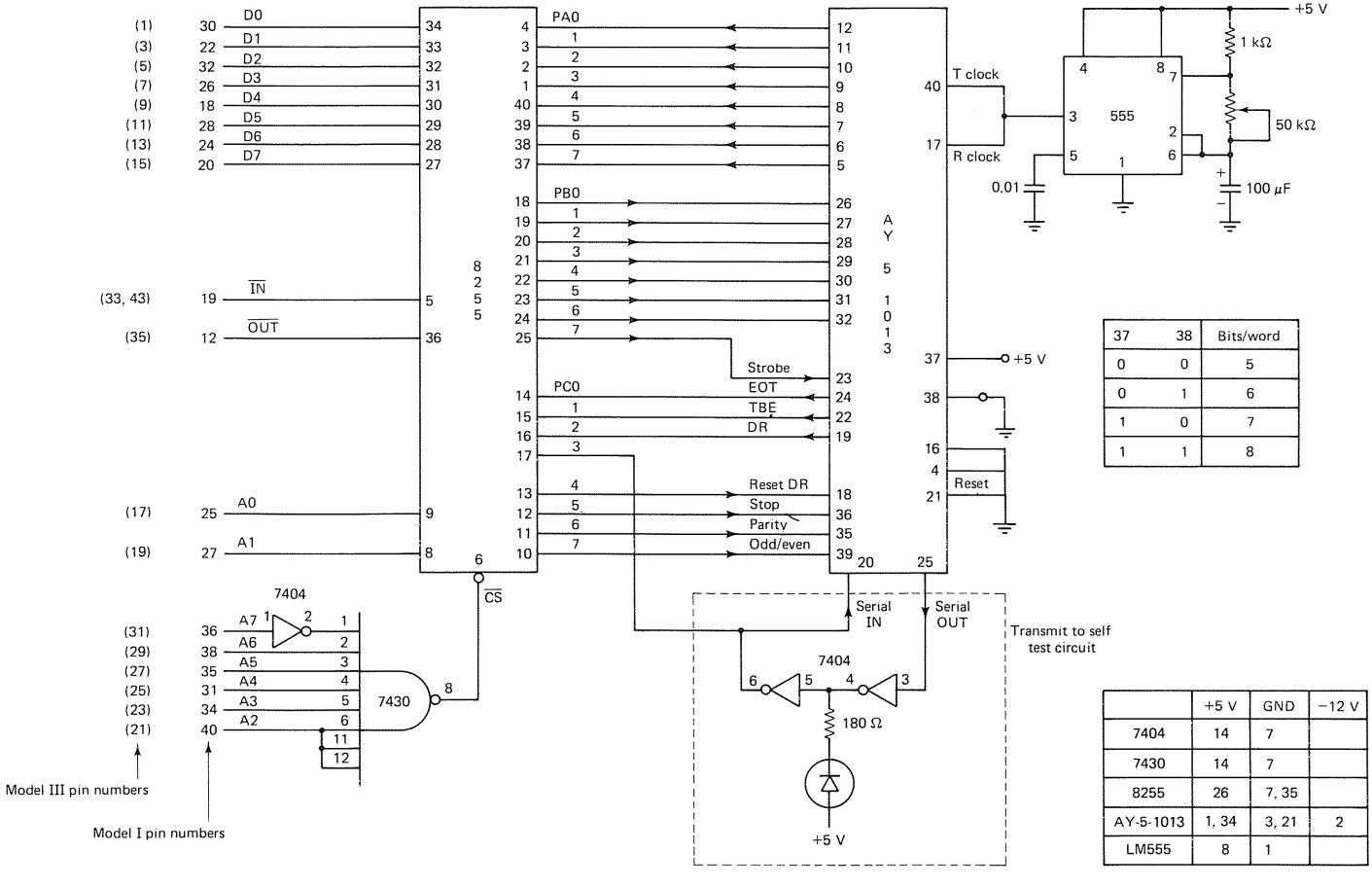
**Example 13-2**

List the four port addresses occupied by the 8255 in Fig. 13-4 and the description of each port.

**Solution** The port addresses are determined by examining the *address decoder*. The $\overline{CS}$ input to the 8255 will go low when the low-order address bus contains 011111XX. This corresponds to ports 124 through $127_{10}$.

| | | |
|---|---|---|
| Port 124 | PA0–PA7 | Input data from UART |
| 125 | PB0–PB6 | Output data to UART |
| 125 | PB7 | Strobe input to UART |
| 126 | PC0–PC3 | Input flags for **EOT**, **TBE**, **DR**, and *SERIAL IN* |
| 126 | PC4–PC7 | Output control for *RESET DR*, *STOP*, *PARITY*, and *ODD/EVEN* |
| 127 | 8255 control | |

The clock (or baud rate generator) in Fig. 13-4 uses a 555 universal timer. Although this circuit is suitable for the purposes of this experiment, a more *stable* clock reference should be selected for a real-world application. The Motorola MC14411 *baud rate generator* is shown in Fig. 13-5. This chip simultaneously generates *14* different baud rates from 75 to 9600 baud using a 1.8432-MHz crystal. The crystal provides stability and accuracy, which is particularly important because the baud rates for the receiving and transmitting UARTs must typically match within *5%*.




| 37 | 38 | Bits/word |
|---|---|---|
| 0 | 0 | 5 |
| 0 | 1 | 6 |
| 1 | 0 | 7 |
| 1 | 1 | 8 |

| | +5 V | GND | −12 V |
|---|---|---|---|
| 7404 | 14 | 7 | |
| 7430 | 14 | 7 | |
| 8255 | 26 | 7, 35 | |
| AY-5-1013 | 1, 34 | 3, 21 | 2 |
| LM555 | 8 | 1 | |

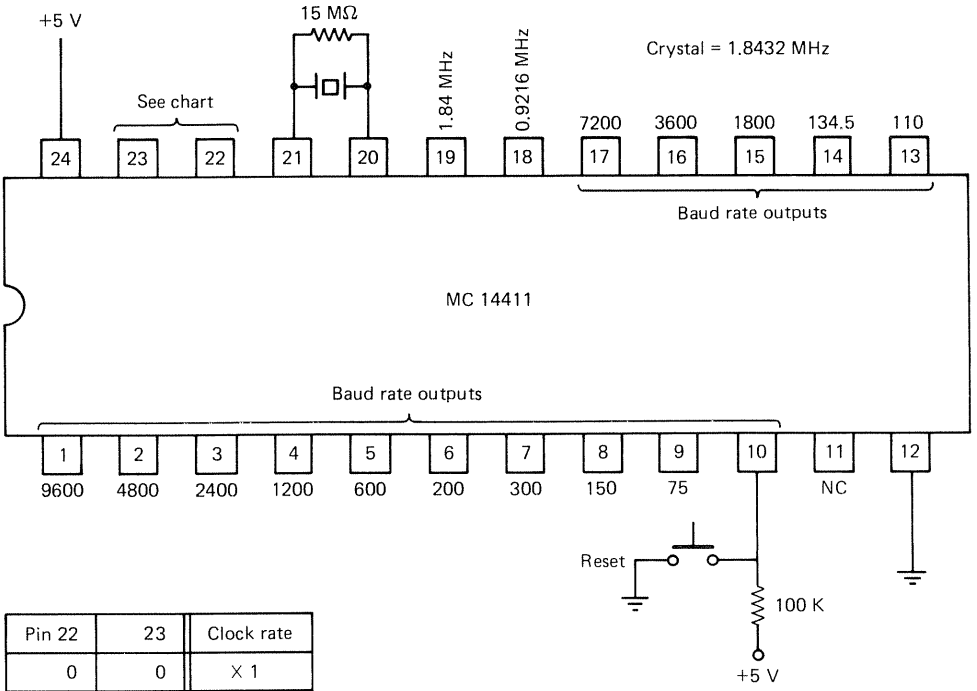
Figure 13-4 TRS-80 interface using the AY-5-1013 UART. Refer to the text for details.

TABLE 13-1 BIT ASSIGNMENTS WHEN PORT C OF THE 8255 IS USED TO CONTROL THE UART IN FIG. 13-4

| 8255 | Type | Name | Description |
|---|---|---|---|
| PC0 | Input | EOT | End of transmission. This line goes high when the full data word (including stop bits) has been transmitted. |
| PC1 | Input | TBE | Transmitter buffer empty. This line goes high when the UART is ready to transmit another data word. |
| PC2 | Input | DR | Data ready. This line goes high when the receiver has received a new data word. |
| PC3 | Input | SERIAL IN | This line follows the input serial data. |
| PC4 | Output | RESET DR | This line is pulsed (⊓⊔⊓) to reset the DR flag after data has been read. |
| PC5 | Output | STOP | This line selects the number of stop bits: a 0 for 1 stop bit and a 1 for 2 stop bits. |
| PC6 | Output | PARITY | This line controls the parity function: a 0 will insert parity, a 1 will inhibit the parity bit. |
| PC7 | Output | ODD/EVEN | If parity is chosen, a 1 on this line selects even parity; a 0, odd parity. |

TABLE 13-2 CONTROL PINS COMMONLY HARDWIRED ON THE AY-5-1013 UART, WITH OUTPUT ERROR FLAGS ALSO LISTED

| Name | Pin | Type | Function |
|---|---|---|---|
| CONTROL | 34 | Input | This input gates control information (pins 35–39) into the UART. It may be hardwired high or strobed ⊔⊓⊔. |
| RESET | 21 | Input | A high level on this pin clears all registers. This pin should be low for normal operation. |
| WORD STATUS | 16 | Input | This pin enables status information on pins 13–15, 19, and 22 when low. When high, these pins are in a tri-state. |
| OVERRUN | 15 | Output | This flag goes high when a new word is received but the *DATA READY* flag has not been reset. |
| FRAMING ERROR | 14 | Output | This flag goes high if no valid stop bit is detected. |
| PARITY ERROR | 13 | Output | This flag goes high if the received word has incorrect parity. |
| DATA RECEIVE ENABLE | 4 | Input | This line is wired low to enable the received data onto the receiver output lines. |




| Pin 22 | 23 | Clock rate |
|---|---|---|
| 0 | 0 | X 1 |
| 0 | 1 | X 8 |
| 1 | 0 | X 16 |
| 1 | 1 | X 64 |

Figure 13-5 The MC14411 generates 14 different baud rates from its 1.8432-MHz crystal time base. Pins 22 and 23 select the output clock rate as 1X, 8X, 16X, or 64X the selected baud rate as required by the UART chosen.

## UART Software

Usually, the UART control software is written in *assembly language* and inserted as a driver routine within the BASIC interpreter itself. In this way, when BASIC is directed to **LIST** a program to the serial printer, this routine is automatically called up and used.

The purpose of this experiment is not to interface a serial port to the BASIC interpreter, but rather to illustrate what a serial port is and how it works. For this reason we will use BASIC rather than assembly language when programming the UART. If you wish to add a serial port to your TRS-80 (usually referred to as an RS-232C port), I refer you to the article by Rod Hallen, "TRS-80 Printer Interfaces: Serial and Parallel Designs," *Kilobaud Microcomputing*, January 1980.

Most of the control options of the UART in Fig. 13-4 are controlled by software. When using this interface the number of bits per data word must first be selected and pins 37 and 38 connected appropriately. For reasons that will become clear in the next section, we will choose 7-bit data words (note that data words of from 5 to 8 bits are possible).

Next, the 8255 must be initialized so that port A is an input, port B an output, PC0 through PC3 are inputs, and PC4 through PC7 are outputs. Referring to the mode definition format in Fig. 5-4, the proper control word is $145_{10}$.

Finally, the high-order bits of port C should be set to the desired UART format. For example, assuming 2 stop bits and even parity and referring to Table 13-1, the appropriate command is

**OUT 126,176**

where bits PC7, PC5, and PC4 are all high (128 + 32 + 16) and port C is located at address 126.

At this point the UART is ready to receive or transmit data. Figure 13-6 illustrates a flowchart to allow the UART to transmit to itself. Note that after configuring the UART the program pauses and waits for the **EOT** (end of transmission) and **TBE** (transmitter buffer empty) flags to go high. This ensures that the UART is not in the middle of inadvertently transmitting a character (perhaps due to the 8255 initialization).

Once the UART has stabilized, the data is output and the strobe line pulsed. Now the program waits for the character to be received, at which time the *DATA READY* flag (**DR**) goes high. The received data can now be input and displayed.

**Example 13-3**

Write a BASIC program to transmit the 7-bit data word 1010101 ($85_{10}$) and then receive it using the "transmit to self" test configuration in Fig. 13-4. Use 2 stop bits and even parity.

**Solution** The program, shown below, follows the flowchart of Fig. 13-6 line by line.

```
 5 CLS
10 OUT 236,16                      :REM MODEL III ONLY
20 OUT 127,145                     :REM INIT PPI
30 OUT 125,128:  OUT 126,176       :REM CONFIGURE UART: STROBE=1,
                                    PC7-PC4=1011
40 IF (INP(126) AND 3) <>3         :REM WAIT FOR EOT AND TBE TO
   THEN 40                          GO HIGH
50 OUT 126,160:  OUT 126,176       :REM RESET THE DR FLAG
60 OUT 125,85:                     :REM STROBE THE UART
   OUT 125,85+128
```

```
 70  IF (INP(126) AND 4) <>4        :REM WAIT FOR DR TO GO HIGH
     THEN 70
 80  Y=INP(124)                     :REM GET THE DATA
 90  PRINT "THE DATA IS: ";Y
100  END
```

The circuit in Fig. 13-4 is set up to transmit to itself, but of course in a real-world application the *SERIAL OUT* and *SERIAL IN* lines would be connected to the *SERIAL IN* and *SERIAL OUT* lines of another UART in the peripheral device.

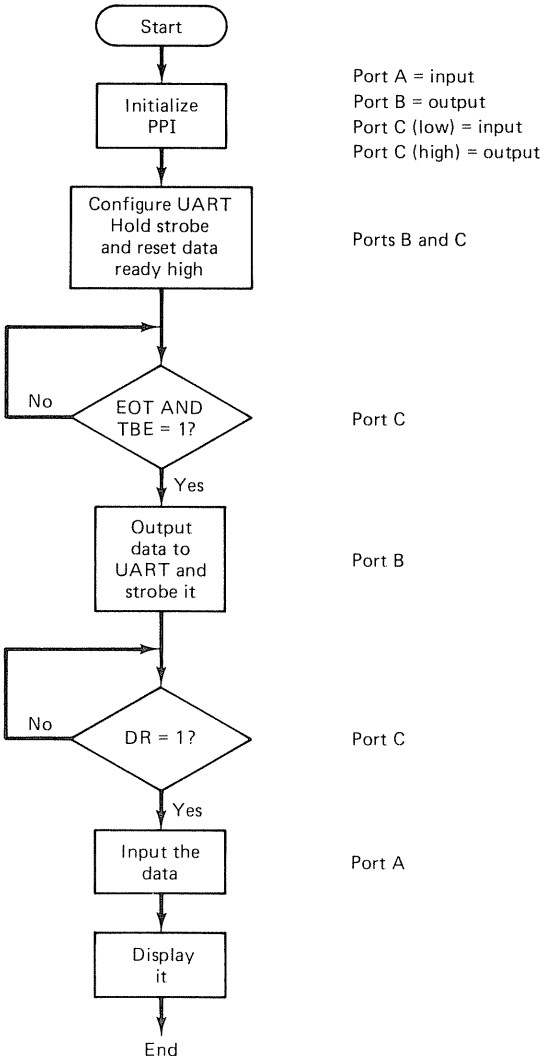



Figure 13-6 UART control flowchart. The various 8255 ports involved are indicated.

Once your computer can transmit and receive serial data, an entire new world of digital communications is open to you. For example, with a serial port and a *modem* you may "call up" other computers over the telephone lines and participate in several of the *time-sharing* networks now available. Or call your stockbroker and get the latest quotations. Perhaps you need information on the undersea life of the *marine iguana* of the Galapagos Islands, The *Dialog Information Retrieval Service* (part of Lockheed Missile and Space Company, Inc.) has some 50 *billion* bytes of information available on line. This corresponds to about 40 million individual bibliographic abstracts and references!*

When using your computer to communicate with other computers (or terminals), the following communications *standards* must be adopted by everyone for compatibility.

1. *The American Standard Code for Information Interchange* (*ASCII*). This is a 7-bit code which defines a particular binary pattern for all the letters of the alphabet, numerals, punctuation marks, and control characters. As long as the ASCII code is used by all manufacturers, when you type a question mark on your keyboard, it will also appear as a question mark on the receiving computer's CRT screen or printer. Table 13-3 is an ASCII code chart included for reference.

2. *The RS-232C serial communications standard.* This standard defines the voltage levels for logic 1's and 0's when communicating serially. You might be surprised to learn that this is *not* TTL compatible; a logic 1 is a voltage that must be more *negative* than −3 V, and a logic 0 must be more *positive* than +3 V. Typical levels used are ±12 V. The reason for the large voltage levels is to provide greater *noise immunity*, particularly when long cable lengths are involved. The RS-232C standard actually goes beyond defining voltage levels. It also specifies a *25-conductor cable* and connector (referred to as a *DB-25*). Although signals are defined for all 25 pins, communications can take place using only *three* of those pins (pin 2, transmit data; pin 3, receive data; and pin 7, signal ground). Figure 13-7 illustrates a typical RS-232C interface between a computer and serial terminal or printer. Note the use of the *MC1488* and *MC1489* to convert the TTL levels to RS-232C levels.

3. The use of *modems.* A modem, or modulator–demodulator, converts the RS-232C voltage levels to two different audio tones. For example, an *originate* modem transmits a 1270-Hz sine wave for a logic 1 and a 1070-Hz sine wave for a logic 0. Modems are used when communicating over long distances via the telephone lines. In situations like this the digital pulses would be hopelessly lost over the long line lengths involved, but the audio

*Stan Miastkowski, "Information Unlimited: The Dialog Information Retrieval System," *BYTE*, June 1981, pp. 88–108.

TABLE 13-3 THE 128 ASCII CHARACTERS AND THEIR DECIMAL EQUIVALENTS

| DEC | | ASCII | DEC | ASCII | DEC | ASCII |
|---|---|---|---|---|---|---|
| 0 | | null | 43 | + | 86 | V |
| 1 | CONTROL | A | 44 | , | 87 | W |
| 2 | ↑ | B | 45 | — | 88 | X |
| 3 | | C | 46 | . | 89 | Y |
| 4 | | D | 47 | / | 90 | Z |
| 5 | | E | 48 | 0 | 91 | [ |
| 6 | | F | 49 | 1 | 92 | \ |
| 7 | | G | 50 | 2 | 93 | ] |
| 8 | | H | 51 | 3 | 94 | ^ |
| 9 | | I | 52 | 4 | 95 | _ |
| 10 | | J | 53 | 5 | 96 | @ |
| 11 | | K | 54 | 6 | 97 | a |
| 12 | | L | 55 | 7 | 98 | b |
| 13 | | M | 56 | 8 | 99 | c |
| 14 | | N | 57 | 9 | 100 | d |
| 15 | | O | 58 | : | 101 | e |
| 16 | | P | 59 | ; | 102 | f |
| 17 | | Q | 60 | < | 103 | g |
| 18 | | R | 61 | = | 104 | h |
| 19 | | S | 62 | > | 105 | i |
| 20 | | T | 63 | ? | 106 | j |
| 21 | | U | 64 | @ | 107 | k |
| 22 | | V | 65 | A | 108 | l |
| 23 | | W | 66 | B | 109 | m |
| 24 | | X | 67 | C | 110 | n |
| 25 | | Y | 68 | D | 111 | o |
| 26 | ↓ | Z | 69 | E | 112 | p |
| 27 | | ESC | 70 | F | 113 | q |
| 28 | | FS | 71 | G | 114 | r |
| 29 | | GS | 72 | H | 115 | s |
| 30 | | RS | 73 | I | 116 | t |
| 31 | | US | 74 | J | 117 | u |
| 32 | | SP | 75 | K | 118 | v |
| 33 | | ! | 76 | L | 119 | w |
| 34 | | " | 77 | M | 120 | x |
| 35 | | # | 78 | N | 121 | y |
| 36 | | $ | 79 | O | 122 | z |
| 37 | | % | 80 | P | 123 | { |
| 38 | | & | 81 | Q | 124 | \| |
| 39 | | ' | 82 | R | 125 | } |
| 40 | | ( | 83 | S | 126 | ~ |
| 41 | | ) | 84 | T | 127 | DEL |
| 42 | | * | 85 | U | | |

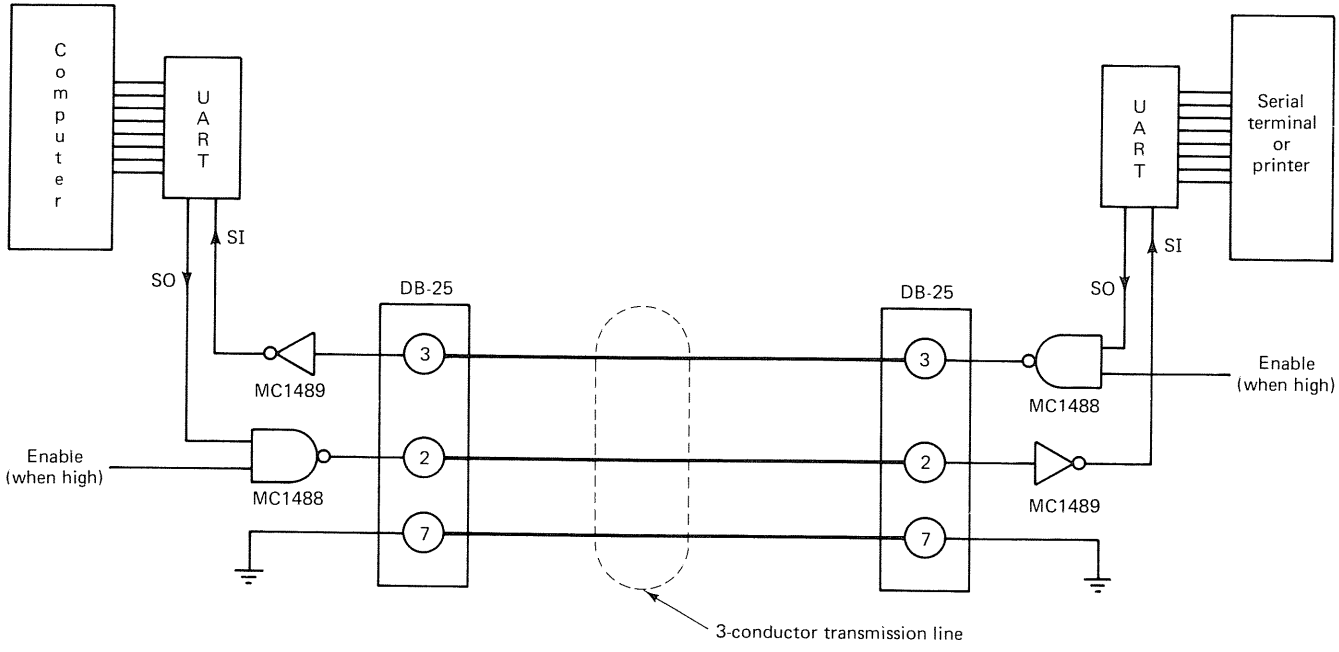



Figure 13-7 Typical RS-232C interface. The MC1488 and 1489 are used to convert the UART's TTL levels to RS-232C levels. The pin numbers shown are for the standard DB-25 connector.

tones pass quite happily over this voice network. When using *full-duplex* operation, transmission and reception at the same time, the receiving modem (called the *answer modem*) uses yet another set of frequencies, 2225 Hz for a 1 and 2025 Hz for a 0. When communicating over the telephone lines the data rate is generally restricted to *300* baud or less.

## PROCEDURE

**Step 1.** If you do not have the 8255 and 7430 decoder wired on your breadboard, do so at this time, referring to Fig. 13-4. Do not wire the UART at this time.

**Step 2.** Refer to Fig. 13-4 and wire the 555 circuit. *Temporarily* connect pin 3 of the 555 to pin 14 (PC0) of the 8255.

**Question 13-1.** Write a BASIC program to initialize the 8255 for the I/O configuration shown in Fig. 13-4 and then cause a count to appear on the CRT screen incrementing each time the 555 output goes from *high* to *low*. Run the program and verify operation.

**Step 3.** With the 555 properly running (it should be adjustable to about a 1-Hz rate), connect the UART to the 8255 ports A, B, and C as shown in Fig. 13-4. Connect *SERIAL OUT* to *SERIAL IN* as shown in the schematic, using two of the spare inverters in the 7404 used for the address decoder.

**Step 4.** Study Example 13-2 for the port assignments and then run the program given in Example 13-3 to test the hardware. Adjust the 555 to its *fastest* rate.

**Question 13-2.** With the 555 running at 1 Hz, how many *total* clock pulses are needed to transmit the full 11 (7 data, 1 start, 1 parity, and 2 stop bits)-bit word?

**Question 13-3.** What is the baud rate corresponding to Question 13-2?

**Step 5.** We may gain some insight into the meaning of the four UART flags connected to PC0 through PC3 by making a slight change to the program given in Example 13-3. Substitute the following three lines for line 70 in that program.

```
65  Y=INP(126) AND 15
70  PRINT Y; "  ";
75  IF Y<>15 THEN 65
```

Now run the program but adjust the 555 clock rate so that the data just fills (without scrolling) your CRT screen (this may require several trial runs to get it right). An example of the display is shown in Fig. 13-8.

**Question 13-4.** Can you explain the significance of this output? You might try again with the data changed to 0. (*Hint:* Convert the numbers to binary.)

**Step 6.** Now change the program you are running so that it first asks you for the data, the number of stop bits, and the type of parity. A solution is provided at the end of this experiment.

**Step 7.** Disconnect the EOT flag (pin 24 of the UART) from PC0 and connect a wire to the 555 output (pin 3) in its place.

*Note.* The program to follow allows full control of the UART for 7-bit ASCII characters, stop bits, and parity. It converts the received data to binary and displays each bit as it is received by counting *16* clock pulses between samples. Finally, it draws the *SERIAL OUT* waveform using the TRS-80 graphics.

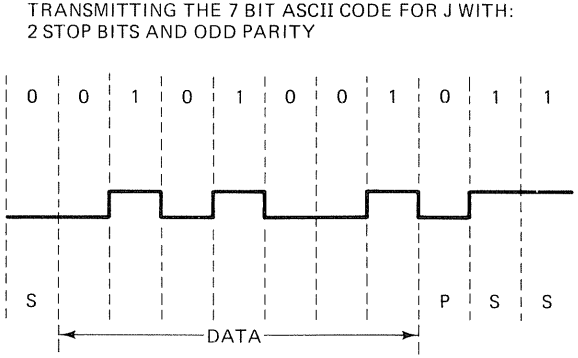
```
9   8   2   2   2   2   2   2   2   2   2   2   2   2   2   2
2   2   2   2   10   10   10   10   10   10   10   10   10
10   10   10   10   10   10   10   10   10   2   2   2   2   2
 2   2   2   2   2   2   2   2   2   2   2   2   2   2   10
10   10   10   10   10   10   10   10   10   10   10   10   10
10   10   10   10   2   2   2   2   2   2   2   2   2   2   2
 2   2   2   2   2   2   2   10   10   10   10   10   10   10
10   10   10   10   10   10   10   10   10   10   10   10   2
2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
2   10   10   10   10   10   10   10   10   10   10   10   10
10   10   10   10   10   10   10   2   2   2   2   2   2   2
2   2   2   2   2   2   2   2   2   2   2   10   10   10   10
10   10   10   10   10   10   14   14   14   14   14   14   14
 14   14   14   14   14   14   14   14   14   14   14   14
14   14   14   14   14   14   15 THE DATA IS: 85
```

(a)

```
9   8   2   2   2   2   2   2   2   2   2   2   2   2   2   2
2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
2   2   2   2   2   10   10   10   10   10   10   10   10   10
 10   14   14   14   14   14   14   14   14   14   14   14   14
14   14   14   14   14   14   14   14   14   41   14   14   14
 14   15 THE DATA IS: 0
```

(b)

Figure 13-8 Output of the program in step 5.




DO YOU WANT TO TRY ANOTHER?

Figure 13-9 Output of the program in step 8.

**Step 8.** Load the program below and run it. See if you can predict the waveform the computer will draw. (*Warning:* It will be necessary to set the 555 to about *half*-speed to achieve proper operation. Otherwise, BASIC does not have enough time to do all the conversions between bit times.) Figure 13-9 illustrates a typical output.

```
  5 CLS:  DIM D(11)
 10 PRINT "THIS PROGRAM WILL
    TRANSMIT AND RECEIVE 7 BIT
    ASCII CHARACTERS"
 20 PRINT:  INPUT "WHAT CHARACTER
    DO YOU WISH TO SEND ";C$
 30 PRINT:  BITS=8:  PC=0                    :REM PC IS UART CONTROL
                                              CODE
 40 INPUT "DO YOU WANT TO INCLUDE
    A PARITY BIT ";P$
 50 IF LEFT$(P$,1)<>"Y" THEN
    PC=PC+64 ELSE BITS=BITS+1
 60 IF LEFT$(P$,1)<>"Y" THEN 80
 65 INPUT "ODD OR EVEN ";P1$
 70 IF LEFT$(P1$,1)="E" THEN
    PC=PC+128
 80 PRINT
 90 INPUT "1 OR 2 STOP BITS ";S
 95 BITS=BITS+S
100 IF S=2 THEN PC=PC+32
```

```
110 CLS
120 OUT 236,16                          :REM MODEL III ONLY
130 OUT 127,145                         :REM INIT PPI
140 PRINT "TRANSMITTING THE 7 BIT
    ACSII CODE FOR ";C$;" WITH: "
150 PRINT S;" STOP BITS ";
160 IF LEFT$(P$,1)="Y" THEN PRINT
    " AND ";P1$;" PARITY"
165 X=270                               :REM FIRST POSITION IN GRAPH
170 FOR J=1 TO 5
180 FOR I=0 TO 33 STEP 3
190 PRINT@(X+I),"!";
200 NEXT I
210 X=X+64
220 NEXT J
230 REM TRANSMIT THE CHARACTER
240 OUT 126,PC+16: OUT 126,PC:          :REM RESET DR AND SET UP
    OUT 126,PC+16                        UART
250 OUT 125,ASC(C$)+128:
    OUT 125,ASC(C$):
    OUT 125,ASC(C$)+128
260 IF INP((126) AND 8)=8 THEN 260      :REM WAIT FOR START BIT
265 K=207                               :REM POSITION OF START BIT
270 PRINT@K,0;
280 GOSUB 500                           :REM WAIT FOR 16 CLOCKS
290 FOR J=1 TO BITS-1
295 K=K+3
300 D(J)=INP(126) AND 8                 :REM SAVE BIT VALUES IN
                                         D(J) ARRAY
305 IF D(J)=8 THEN PRINT@K,1; ELSE      :REM PRINT DATA ON GRAPH
    PRINT@K,0;
310 GOSUB 500                           :REM WAIT FOR 16 CLOCKS
320 NEXT J
330 REM NOW DRAW THE WAVEFORMS
335 PRINT@398, "";                      :REM POSITION CURSOR
340 GOSUB 1000                          :REM DRAW THE START BIT
345 FOR J=1 TO BITS-1: IF D(J)=8        :REM CONVERT SAMPLES TO 1's
    THEN D(J)=1                          AND 0's
347 NEXT J
350 IF D(1)=0 THEN GOSUB 1000 ELSE      :REM DRAW 1ST DATA BIT
    GOSUB 2000
360 FOR J=2 TO BITS-1
365 L=(D(J-1)*2+D(J)+1)                 :REM CALCULATE WHICH SUB IS
                                         NEEDED
370 ON L GOSUB 1000, 2000, 3000, 4000
380 NEXT J
390 PRINT@591,
    "S <------ D A T A ------>";
```

```
 400  IF LEFT$(P$,1)="Y" THEN
      PRINT " P";
 410  PRINT "  S";
 420  IF S=2 THEN PRINT "  S"
 430  PRINT:  PRINT:  PRINT
 440  INPUT "DO YOU WANT TO TRY
      ANOTHER ";A$
 450  IF LEFT$(A$,1)="Y" THEN CLS:
      GOTO 10
 460  END
 500  C=0
 510  IF (INP(126) AND 1)=0 THEN 510
 520  IF (INP(126) AND 1)=1 THEN 520
 530  C=C+1
 540  IF C<>16 THEN 510
 550  RETURN
1000  PRINT CHR$(176) ;CHR$(176);          :REM 0 TO 0
      CHR$(176);:  RETURN
2000  PRINT CHR$(151) ;CHR$(131);          :REM 0 TO 1
      CHR$(131);:  RETURN
3000  PRINT CHR$(181) ;CHR$(176);          :REM 1 TO 0
      CHR$(176);:  RETURN
4000  PRINT CHR$(131) ;CHR(131);           :REM 1 TO 1
      CHRS$(131);:  RETURN
```

**Step 9.** Reread the objectives as stated at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

*Note.* The 8255 portion of this interface will be used once more in Experiment 14.

## SOLUTIONS TO QUESTIONS

**13-1.**

```
  5  CLS
 10  OUT 236,16                          :REM MODEL III ONLY
 20  OUT 127,145                         :REM INIT PPI
 30  IF (INP(126) AND 1)=0 THEN 30       :REM WAIT FOR PC0 TO BE HIGH
 40  IF (INP(126) AND 1)=1 THEN 40       :REM WAIT FOR HIGH TO LOW
                                          TRANSITION
 50  C=C+1
 60  PRINT@484,C
 70  GOTO 30
```

*Note.* As you adjust $R_{50K}$ of the 555 you should see the count rate change correspondingly. Eventually, the rate will be *too fast* for the computer to follow and the count will appear to *jump* instead of increment sequentially.

**13-2.** The UART requires 16 clock pulses per bit. For an 11-bit word, 16 × 11 = 176 clock pulses are required. It will therefore take nearly *3 minutes* to transmit the full word with a 1-Hz clock!

**13-3.** The baud rate represents bits per second. With a 1-Hz clock it will take 16 s per bit. The baud rate is 1/16 or 0.0625 baud (very slow!).

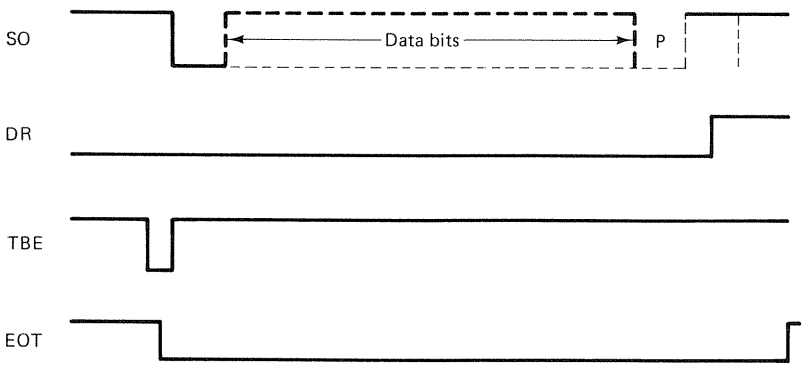
**13-4.** Figure 13-8 illustrates the typical output. Depending on the speed of your 555, you may have a few more or less lines on your CRT.

Figure 13-10 explains the significance of the decimal output codes. This in-

| Comment | SO | DR | TBE | EOT | Decimal |
|---|---|---|---|---|---|
| Transmission not yet begun | 1 | 0 | 0 | 1 | 9 |
| Transmission begins, EOT = 0 | 1 | 0 | 0 | 0 | 8 |
| Start bit occurs, SO = 0 | 0 | 0 | 1 | 0 | 2 |
| Transmitting a 1 | 1 | 0 | 1 | 0 | 10 |
| All data bits received, DR = 1 | 1 | 1 | 1 | 0 | 14 |
| EOT = 1 signifying all bits transmitted | 1 | 1 | 1 | 1 | 15 |

Note: DR = data ready
EOT = end of transmission
SO = serial out
TBE = transmitter buffer empty

(a)




(b)

Figure 13-10 The meaning of the decimal codes displayed in Fig. 13-8 is indicated in (a). In (b) this information is shown as digital waveforms.

formation in turn allows an interpretation of the data in Fig. 13-8. Referring to Fig. 13-8, the **Y=INP(126) AND 15** command catches the UART before transmission begins (code = 9, 1001). Next, the UART begins to transmit the data word and its EOT flag goes low (code = 8, 1000). The start bit, a 0, is next (code = 2, 0010). The data bits now follow with a 1 represented by the code 10 (1010) and a 0 represented by a 2 (0010). Notice that Fig. 13-8a allows us to see "how long" a bit time is. For example, the start bit lasts for *18* samples (count the first set of 2s in Fig. 13-8a). Now, by counting carefully, you will see that the **DR** flag goes high (code = 14, 1110) in the *middle* of the first stop bit. Finally, the **EOT** flag goes high (code = 15, 1111) after both stop bits have been sent.

All this information can then be used to draw the waveforms shown in Fig. 13-10b.

STEP 6 (solution)

```
  5  CLS:  PC=0                          :REM PC IS UART CONTROL CODE
  7  GOSUB 200                           :REM THIS IS A PATCH TO STEP 5
 10  OUT 236,16                          :REM MODEL III ONLY
 20  OUT 127,145:                        :REM INIT PPI
 30  OUT 125,128:                        :REM CONFIGURE UART, STROBE=1
     OUT 126,PC+16
 40  IF (INP(126) AND 3) <>3             :REM WAIT FOR EOT AND TBE=1
     THEN 40
 50  OUT 126,PC:  OUT 126,PC+16          :REM RESET DR FLAG
 60  OUT 125,D:  OUT 125,D+128           :REM STROBE THE UART
 70  Y=INP(126) AND 15                   :REM READ STATUS BITS
 70  PRINT Y;" ";                        :REM PRINT THEM SEQUENTIALLY
 75  IF Y<>15 THEN 70                    :REM CONTINUE UNTIL EOT
 80  Y=INP(124)                          :REM NOW GET DATA
 90  PRINT "THE DATA IS: ";Y
100  END
200  INPUT "WHAT DATA DO
     YOU WISH TO TRANSMIT
     (0-127) ";D
210  INPUT "ONE OR TWO STOP
     BITS ";S
220  INPUT "EVEN OR ODD
     PARITY ";P$
230  IF S=2 THEN PC=PC+32
240  IF LEFT$(P$,1)="E" THEN
     PC=PC+128
250  CLS
260  RETURN
```

# EXPERIMENT 14

## INTERFACING A PROGRAMMABLE SOUND GENERATOR

### OVERVIEW

In this experiment you will interface a General Instruments programmable sound generator (*PSG*) to the TRS-80 expansion bus using an 8255 PPI chip. BASIC programs will be developed to test all internal registers and modes of operation for the PSG. A program that converts the TRS-80 and keyboard into an electronic organ with memory is given.

### OBJECTIVES

The key points to be learned from this experiment are:

1. The AY-3-8910 is designed for *computer control* and does not require external resistors or capacitors to achieve the various sound effects.
2. The PSG has two modes of operation: *fixed amplitude* or *envelope generator* control.
3. The data bus and address bus of the PSG are *multiplexed*, requiring the interface to follow a specific *sequence* when writing data to the chip.
4. In some cases the execution of BASIC commands is too *slow* for strobing I/O devices, and external hardware must be designed into the interface to develop the proper strobe pulse widths.

### PARTS LIST

1 AY-3-8910 programmable sound generator (Digital Research: Computers, P.O. Box 401565, Garland, TX 75040)

1 8255 programmable peripheral interface (JAMECO INS 8255)
1 4013 dual D flip-flop (CMOS)
1 4069 hex inverter (CMOS)
1 7400 quad NAND gate
1 7404 hex inverter
1 7430 8-input NAND gate
1 74121 one-shot
1 LM386 audio amplifier (Radio Shack 276-1731)
1 10-MΩ resistor (brown–black–blue)
1 300-Ω resistor (orange–black–brown)
2 4.7-kΩ resistor (yellow–violet–red)
1 1-kΩ resistor (brown–black–red)
1 470-Ω resistor (yellow–violet–brown)
1 20-pF capacitor
1 0.001-μF capacitor
1 0.1-μF capacitor
1 300 to 500-pF capacitor
1 100-μF or greater capacitor
2 10-μF capacitor
1 3.579545-MHz crystal (Radio Shack 272-1310)
1 loudspeaker

## DISCUSSION

### Sound Generators

Computers have been used to generate music and various sound effects since their inception. Early microprocessors were programmed to execute specific time-delay loops. The cyclic pattern established on the address bus as these loops were executed radiated to nearby transistor radios and music could be created.

More recent techniques have involved turning on and off oscillator circuits pretuned to produce specific notes. This method is illustrated in Fig. 14-1. A computer output port is then used to activate the particular notes desired. Chips such as the Mostek *MK50240N* top octave generator simplify this design by incorporating all the circuitry needed to produce one full octave on the equal-tempered scale in a single IC package. This IC is illustrated in Fig. 14-2.

The most recent developments have been to produce ICs capable of *complex* sound generation. This includes tone, noise, and envelope control

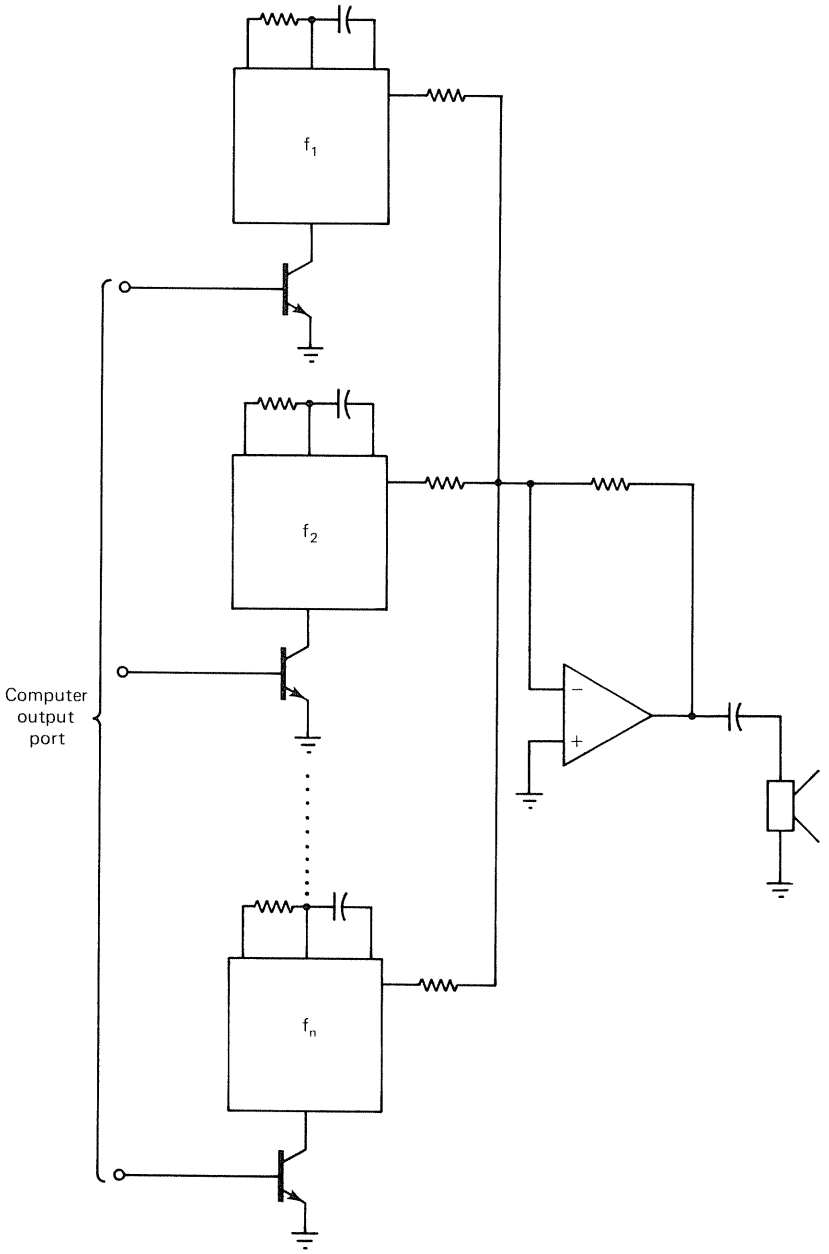



Figure 14-1 A computer output port can be used to turn on or off several pretuned oscillator circuits. The op-amp combines their output and drives the speaker.

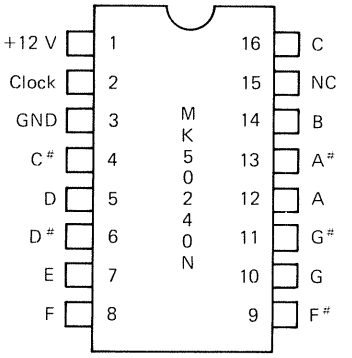



Figure 14-2 The Mostek MK50240N develops the 13 frequencies equivalent to the equal-tempered scale from a 2-MHz clock input.

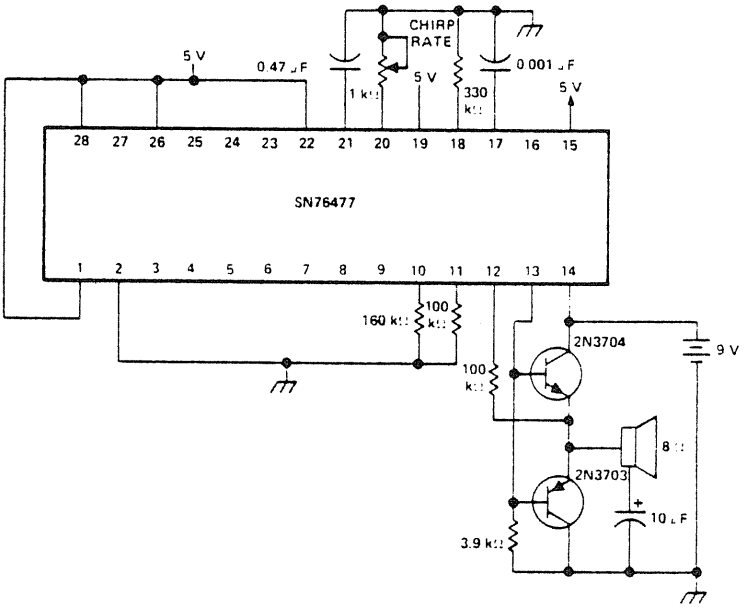
of the waveshape attack and decay times. With these ICs it is possible to simulate the sound of a train whistle, police siren, gunshot, explosion, and thousands of other sound effects.

Two types of complex sound generators are currently available. In the first, external capacitors and resistors establish basic circuit time constants and control the various waveshapes and tones. This type of circuit is generally *hardwired* to produce a particular sound or range of sounds. For example, the circuit in Fig. 14-3 produces the sound of a bird chirping, but by lowering the capacitance at pin 17 a "barking dog" effect is produced. Still other sounds can be produced by rewiring the circuit altogether. For example, the wiring needed to produce a gunshot or explosion is illustrated in Fig. 14-4.

The chip shown in both circuits is the Texas Instruments *SN76477*. This IC incorporates both digital and analog circuitry in one IC and includes a noise generator, voltage-controlled oscillator, super-low frequency oscillator, noise filter, mixer, and envelope control circuit. Its particular function is programmed by wiring control pins high or low and connecting various resistor–capacitor combinations to the other pins. Although the SN76477 can be controlled by a microcomputer, it is a bit awkward and requires a substantial number of analog parts (resistors and capacitors).

The *SN76488* incorporates an on-chip audio amplifier capable of driving a small speaker. This eliminates the need for the two-transistor amplifier in Figs. 14-3 and 14-4.

A second type of complex sound generator is characterized by the General Instruments *AY-3-8910* programmable sound generator shown in Fig. 14-5a and the TI *SN76489A* shown in Fig. 14-5b. These chips, much like the Intel *8255* programmable peripheral interface, are meant to be controlled by a microcomputer. All of their functions are controlled via an 8-bit bidirectional data bus connected to a microprocessor. No rewiring is necessary to change the sounds produced. This is because a new control word transmitted by the microprocessor accomplishes this function. In this




For barking dog, the capacitor at pin 17 is changed to 15 pF to increase the frequency of the VCO.

Figure 14-3 Using the TI SN76477 to simulate a chirping bird. (Courtesy of Texas Instruments, Inc.)

experiment we will concentrate on the GI AY-3-8910 and its interface to the TRS-80 expansion bus.

## The GI AY-3-8910

The General Instruments AY-3-8910 programmable sound generator (PSG) is an extremely powerful integrated circuit. Using a standard "color TV" crystal (3.58 MHz) for its time base, each of its three output channels can be programmed to produce a tone in the range *55 Hz to 224 kHz* in 4096 discrete steps! The *volume* of each channel is also programmable and 16 different levels are possible. Individual channels may have different tones, allowing *cords* and *resonance* effects to be produced. The chip also contains a *noise generator* useful for simulating explosions, gunshots, and percussion instruments. The output envelope or waveshape can also be controlled to allow variable attack and decay periods. For example, a ringing bell is simulated by an abrupt attack time but a longer decay period.

In addition to its sound capabilities, the AY-3-8910 features two programmable I/O ports that can be used similar to the I/O ports in the 8255. The *AY-3-8912* is identical but features a single I/O port.

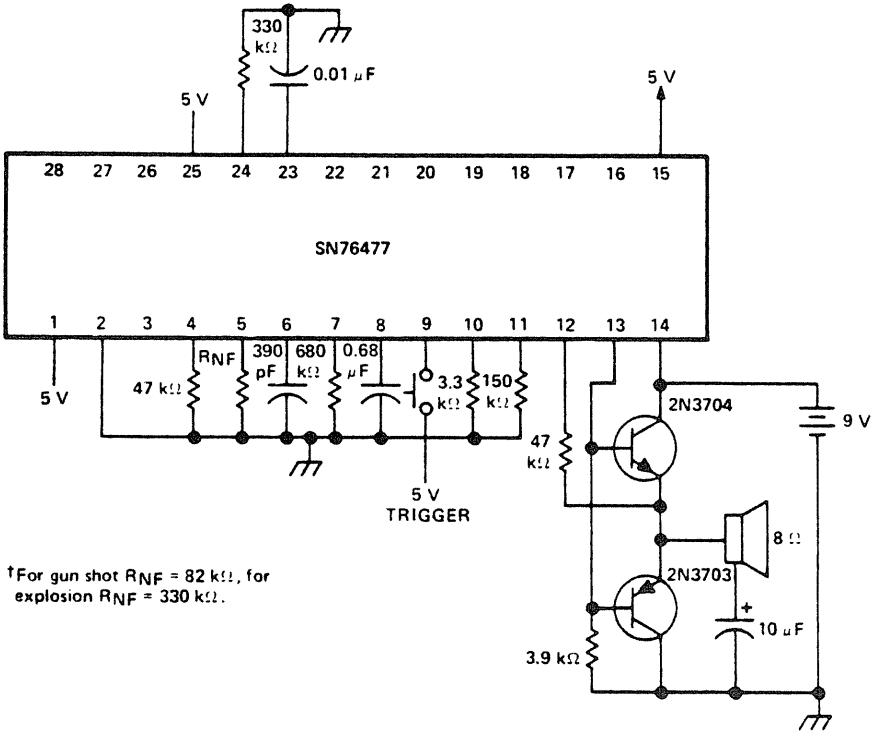


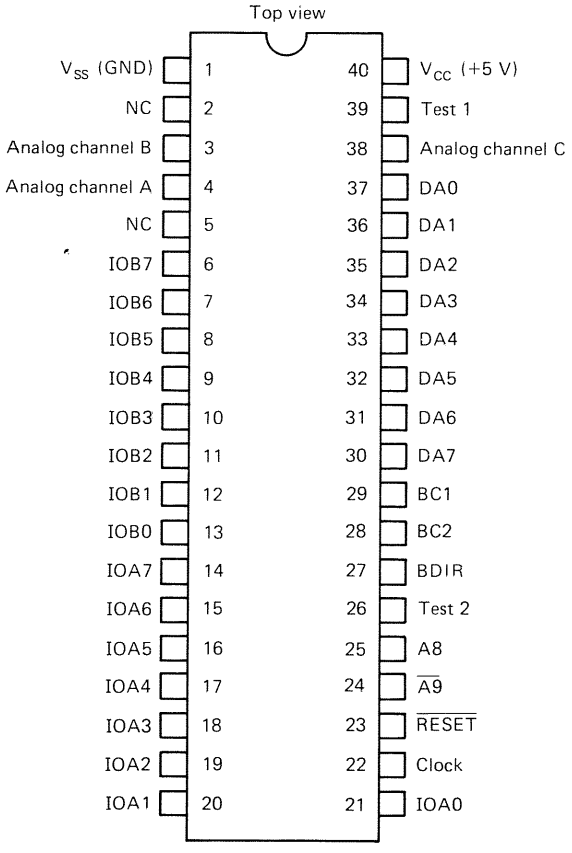

Figure 14-4 The circuit in Fig. 14-3 is rewired to produce a gunshot or explosion effect. Note the number of external resistors and capacitors required. (Courtesy of Texas Instruments, Inc.)

Interfacing the AY-3-8910 to the TRS-80

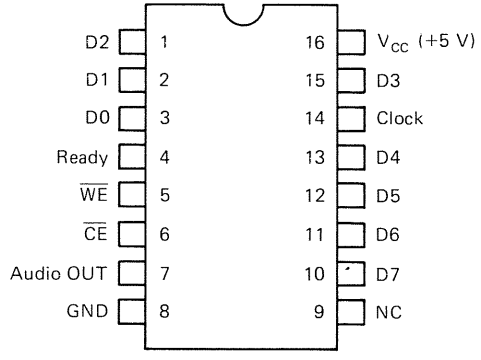
*Note.* The AY-3-8910 is a very complex IC. In explaining its operation we will first treat it as a "black box" and show how it is interfaced to the TRS-80. After this is understood, we will study its internal register array and how to program these registers.

Figure 14-6 illustrates an interface circuit between the TRS-80 and the PSG chip. There are several important aspects to this interface.

1. *Clock generator.* The PSG must receive a TTL-level clock signal at pin 22 with a frequency of between 1 and 2 MHz. The circuit shown derives its time base from a 3.58-MHz TV crystal. The 4013 CMOS flip-flop provides a divide-by-2 function. The frequency applied to pin 22 is therefore 1.79 MHz.
2. PSG *data bus.* All communication with the PSG is through an 8-bit bi-directional data bus at pins 30 through 37. Port A of the 8255 is pro-

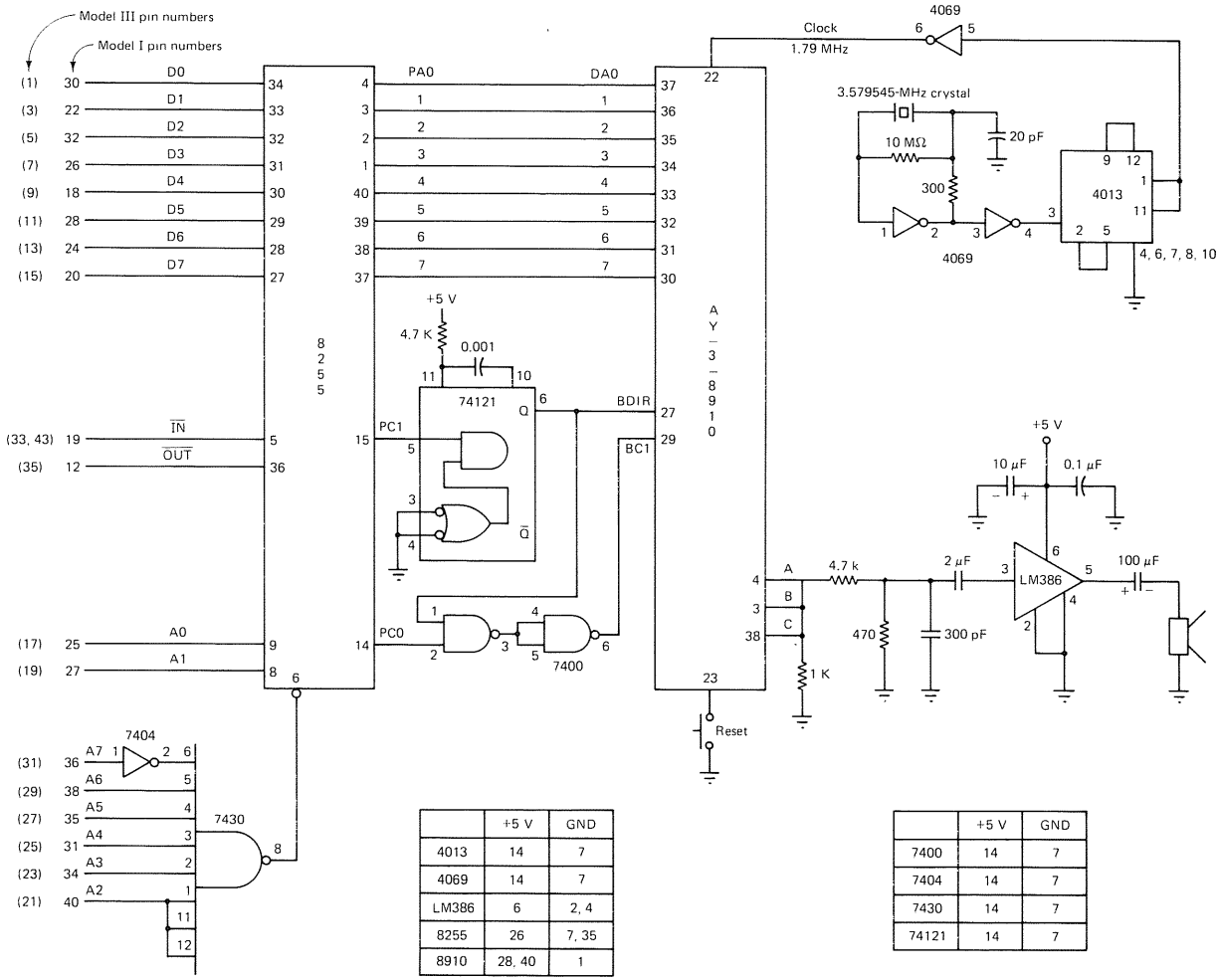



(a) AY-3-8910




(b) SN76489A

Figure 14-5 Pin diagrams for the (a) AY-3-8910 and (b) SN76489A programmable complex sound generators.




| | +5 V | GND |
|---|---|---|
| 4013 | 14 | 7 |
| 4069 | 14 | 7 |
| LM386 | 6 | 2, 4 |
| 8255 | 26 | 7, 35 |
| 8910 | 28, 40 | 1 |

| | +5 V | GND |
|---|---|---|
| 7400 | 14 | 7 |
| 7404 | 14 | 7 |
| 7430 | 14 | 7 |
| 74121 | 14 | 7 |

Figure 14-6 Interface circuit between the TRS-80 expansion bus and the AY-3-8910 programmable sound generator. The 74121 one-shot is necessary to meet the write data pulse width specification.

grammed as an output port and data is output to the PSG with the command

OUT 124,DATA

3. PSG *control bus.* The control bus of the PSG determines the mode of operation of the chip and how it will interpret data on its data bus. Figure 14-7 lists the four possible modes. Normally, the chip is held in the *inactive* mode. When writing data, the address of the internal register desired must first be output and the control bus brought momentarily to the *latch address* state. Any subsequent data writes or reads will now be directed to or from this latched register. If it is desired to read or write from a different register, a new address must be output and the control bus momentarily sequenced through the latch address state again.

Timing diagrams for the *latch address*, *write data*, and *read data* modes are illustrated in Fig. 14-8. When in the write mode care must be taken not to exceed the write data width ($t_{DW}$), which is specified as 10 $\mu$s maximum by the manufacturer. This is the purpose of the *one-shot* connected to PC1 in Fig. 14-6. When PC1 goes high, the one-shot output will pulse high for approximately 3 $\mu$s (within the $t_{DW}$ specification). Using BASIC it is not possible to pulse the PC1 line from low to high to low in less than 3 to 5 ms (3000 to 5000 $\mu$s) and for this reason the one-shot is required.

**Example 14-1**

What BASIC commands must be given to cause the PSG control bus to sequence through each of the four modes listed in Fig. 14-7?

**Solution** Because the 8255 is I/O mapped at ports 124 through 127 and the PSG control bus is connected to port C, all commands are of the form

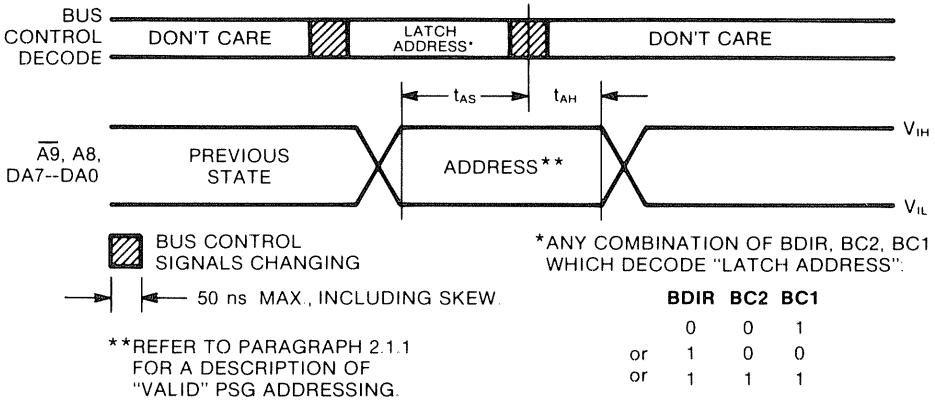
OUT 126,XX

where port C corresponds to address 126.

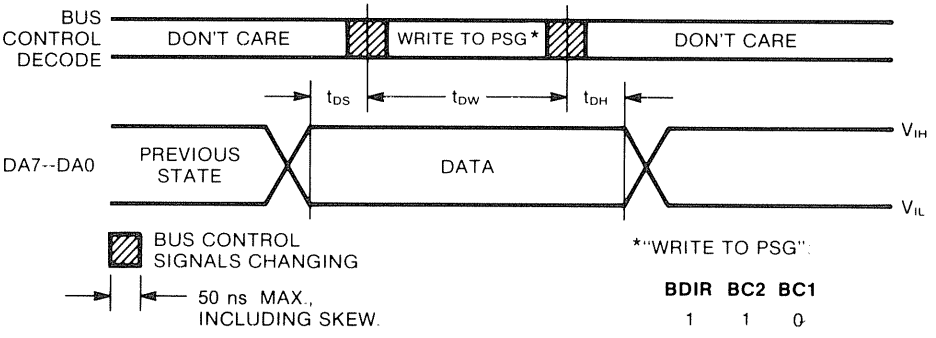
| Name | BDIR (pin 27) | BC1 (pin 29) | Function |
|---|---|---|---|
| Inactive | 0 | 0 | Hold. Chip should normally be held in this state. |
| Read | 0 | 1 | Read data from the PSG. |
| Write | 1 | 0 | Write data to the PSG. |
| Latch address | 1 | 1 | The data on the data bus should be latched and interpreted as an internal register address. |

Figure 14-7 The four basic functions of the PSG as determined by the binary pattern on the control bus.

## LATCH ADDRESS TIMING




## WRITE DATA TIMING




## READ DATA TIMING

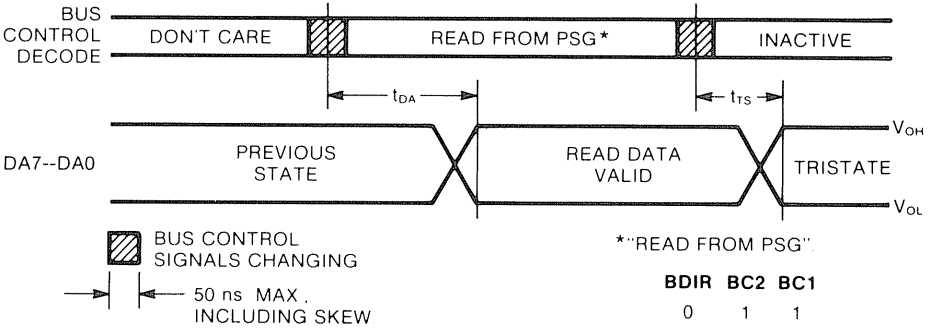



Figure 14-8 Timing relationships for the AY-3-8910 PSG. (Courtesy of General Instruments, Microelectronics Division.)

The *inactive* state is entered with the command

```
OUT 126,0          :REM INACTIVE
```

The *write* mode requires the command

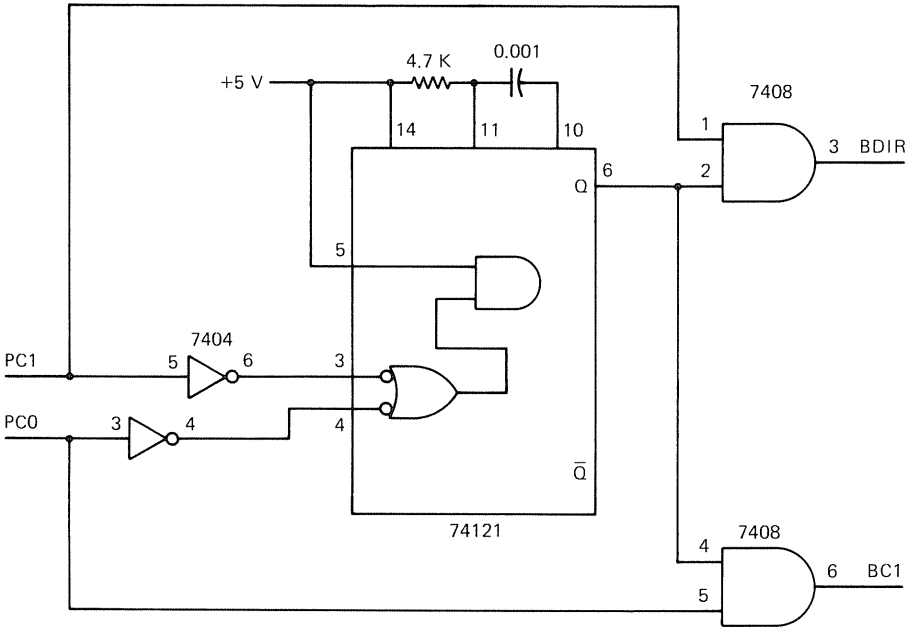
```
OUT 126,2          :REM WRITE
```

The *latch address* mode requires the command

```
OUT 126,3          :REM LATCH ADDRESS
```

Notice that for this last case PC0 enables the one-shot pulse to pass to BC1 through the 7400 and *both* pins 27 and 29 pulse high as required.

It would appear that the *read* mode could be entered with the command OUT 126,1. However, this will cause PC1 to be low, disabling the one-shot and preventing pin 29 from pulsing high as required. The circuit as shown cannot be used in the read mode. Figure 14-9 illustrates the modification needed to enable




| IC | +5 V | GND |
|---|---|---|
| 7404 | 14 | 7 |
| 74121 | 14 | 7 |
| 7408 | 14 | 7 |

Figure 14-9 Modifications to the interface circuit in Fig. 14-6 to allow all four modes of operation on the control bus.

this mode of operation. In this experiment we will only be testing the sound generator portion of the PSG, not its built-in I/O ports. Therefore, the read mode is not necessary.

## PSG Software

Controlling the AY-3-8910 from BASIC requires that a specific *sequence* be followed to address and write properly into a desired register. Refer to the waveforms in Fig. 14-8 for the following sequence.

1. Output the register address.
2. Set the control lines to *latch address* (BDIR = 1, BC1 = 1).
3. Set the control lines to inactive (BDIR = 0, BC1 = 0).
4. Output the data for the selected register.
5. Set the control lines to *write data* (BDIR = 1, BC1 = 0).
6. Set the control lines to *inactive* (BDIR = 0, BC1 = 0).

Each time new data is to be written to the PSG, the foregoing sequence must be followed. In addition, the control lines must not stay in the write data state for longer than 10 $\mu$s (refer to the discussion on the control bus in the preceding section).

**Example 14-2**

Write a BASIC routine that will input the desired register and data from the keyboard and output this information to the PSG using the proper sequence. Assume the hardware configuration shown in Fig. 14-6.

**Solution**

```
10 CLS
20 OUT 236,16                                  :REM MODEL III ONLY
30 OUT 127,128                                 :REM INIT PPI AND MAKE
                                                ALL PORTS OUTPUTS
40 INPUT "REGISTER AND DATA ";R,D
50 OUT 124,R: OUT 126,3: OUT 126,0             :REM LATCH ADDRESS
60 OUT 124,D: OUT 126,2: OUT 126,0             :REM WRITE DATA
70 GOTO 40
```

Notice how the program follows the six steps listed previously. The PSG data bus is connected to port 124 (8255 port A) and the control bus is controlled by bits PC0 and PC1 of port 126 (8255 port C).

Now that the hardware is functional, all that remains is to determine what data to write to which register to make the PSG sound off! This is the hard part, but also the fun part!

Table 14-1 lists the functions of all 16 registers within the PSG. *Seven* distinct functions exist (actually only six functions control sound generation, as registers 14 and 15 correspond to the two programmable I/O ports).

As Table 14-1 indicates, the PSG can be operated in two specific modes. In the first, called the *envelope* mode, register *13* controls the output amplitude and therefore *waveshape*. Figure 14-10 illustrates the various waveshapes possible. Notice that only 4 bits of this register are used (B0 through B3). Control of the waveshape allows for various sound effects. For example, writing a 0 to register 13 with the tone generator enabled and a long decay time will simulate a ringing bell.

The second mode of operation is called the *fixed-amplitude* mode. In this mode, the amplitude of each channel is set by registers 8 through 10 to one of *16* levels. No control of the waveshape is possible.

With either mode of operation, the chip can be enabled to produce a

TABLE 14-1 THE 16 REGISTERS OF THE AY-3-8910 PSG[a]

| Register numbers | Function | Active in: Envelope mode | Active in: Fixed-amplitude mode |
|---|---|---|---|
| 0–5 | Controls the tone on channels A, B, and C. Can be set between 55 Hz and and 224 kHz. | Yes | Yes |
| 6 | Noise generator frequency. This frequency can be varied between 3.6 and 112 kHz. | Yes | Yes |
| 7 | Mixer control. This register allows any combination of channels to be enabled for noise and/or tones. | Yes | Yes |
| 8–10 | Amplitude control. The amplitude of each channel can be set to one of 16 levels. | No | Yes |
| 11–12 | Envelope period control. These registers control the frequency or period of the output waveform in the envelope mode. They allow control of the attack and decay times. | Yes | No |
| 13 | Envelope shape. The actual envelope shape is set by the value in this register. Refer to Fig. 14-10 for specific waveshapes. | Yes | No |
| 14–15 | I/O port data store. | Yes | Yes |

[a]Frequencies specified assume a 1.79-MHz clock frequency.

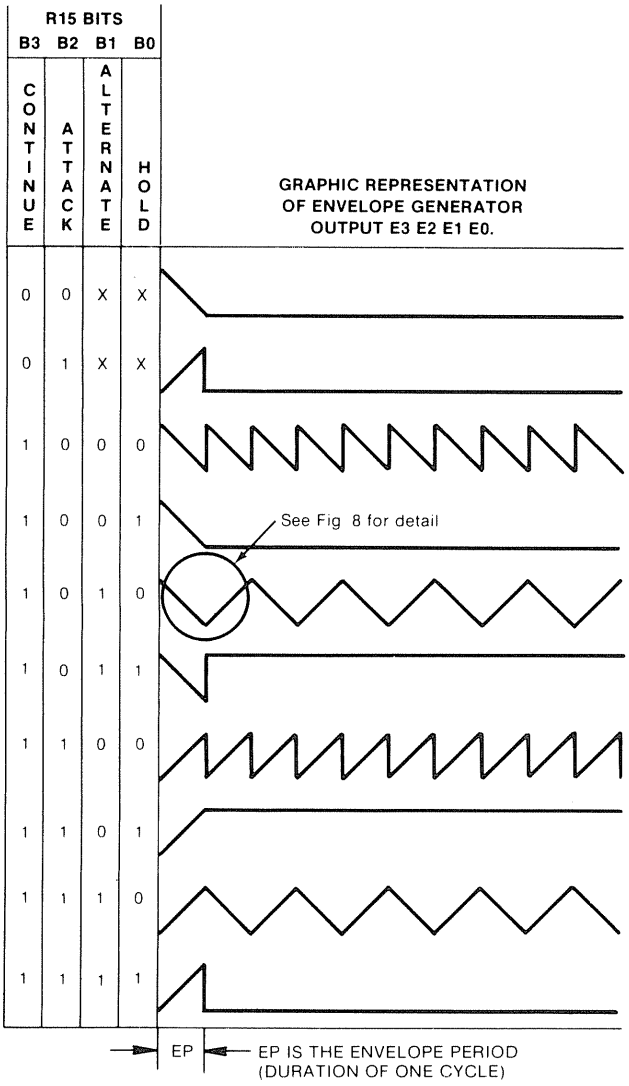



Figure 14-10 Ten separate waveshapes are possible when operating the PSG in the envelope control mode. Bits B0 through B3 of register 13 control this selection. (Courtesy of General Instruments, Microelectronics Division.)

| REGISTER \ BIT | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|---|---|
| R0 | Channel A Tone Period | 8-BIT Fine Tune A | | | | | | | |
| R1 | | | | | | 4-BIT Coarse Tune A | | | |
| R2 | Channel B Tone Period | 8-BIT Fine Tune B | | | | | | | |
| R3 | | | | | | 4-BIT Coarse Tune B | | | |
| R4 | Channel C Tone Period | 8-BIT Fine Tune C | | | | | | | |
| R5 | | | | | | 4-BIT Coarse Tune C | | | |
| R6 | Noise Period | | | | 5-BIT Period Control | | | | |
| R7 | $\overline{\text{Enable}}$ | $\overline{\text{IN}}$/OUT | | $\overline{\text{Noise}}$ | | | $\overline{\text{Tone}}$ | | |
| | | IOB | IOA | C | B | A | C | B | A |
| R8 | Channel A Amplitude | | | | M* | L3 | L2 | L1 | L0 |
| R9 | Channel B Amplitude | | | | M* | L3 | L2 | L1 | L0 |
| R10 | Channel C Amplitude | | | | M* | L3 | L2 | L1 | L0 |
| R11 | Envelope Period | 8-BIT Fine Tune E | | | | | | | |
| R12 | | 8-BIT Coarse Tune E | | | | | | | |
| R13 | Envelope Shape/Cycle | | | | | CONT. | ATT. | ALT. | HOLD |
| R14 | I/O Port A Data Store | 8-BIT PARALLEL I/O on Port A | | | | | | | |
| R15 | I/O Port B Data Store | 8-BIT PARALLEL I/O Port B | | | | | | | |

Tone period: $f_{tone} = \frac{f_{clock}}{16TP_{10}}$

$TP_{10} = 256CT + FT$
CT = coarse tune register (1, 3, 5)
FT = fine tune register (0, 2, 4)

Noise period: $f_N = \frac{f_{clock}}{16NP_{10}}$

$NP_{10}$ = decimal equivalent of register 6

Envelope period: $f_E = \frac{f_{clock}}{256EP_{10}}$

$EP_{10} = 256CT + FT$
CT = coarse tune register (12)
FT = fine tune register (11)

*Mode bit: M = 1 for fixed amplitude mode
M = 0 for envelope generator mode

Figure 14-11 Summary of all 16 PSG register functions. (Courtesy of General Instruments, Microelectronics Division.)

*tone* or a *noise* signal or both simultaneously on any of its three channels. Register 7 controls the mixing of these combinations.

Although we could try to describe the operation of each individual register, the best way to understand the capabilities of this IC is to assemble the test circuit in Fig. 14-6 and experiment with it yourself. Figure 14-11 summarizes the function of all 16 registers and is obtained from the General Instruments *Programmable Sound Generator Data Manual.* This manual is a necessity for working with this chip and can be obtained by writing to General Instruments Corp., Microelectronics Division, 600 West John Street, Hicksville, NY 11802.

## PROCEDURE

**Step 1.** Refer to Fig. 14-6 and carefully wire this circuit on your breadboard. This is a complex circuit requiring *nine* ICs, so take your time to be sure that each connection is correct. If you have an audio amplifier, the LM386 portion of the interface can be skipped and the output from pins 3, 4, and 38 across the 1-kΩ resistor connected to your amplifier. Take care when handling the AY-3-8910, as it is a *static-sensitive* MOS device.

**Step 2.** Test your hardware by running the program given in Example 14-2. Type the following responses to the "REGISTER AND DATA" prompt:

1,1 Set the coarse tone for channel A to 1.
7,62 Enable tone on channel A only.
8,10 Set the volume for channel A to 10.

After typing 8,10 you should hear a low-pitched tone. Try changing the volume by writing new data to register 8. Change the tone by writing to register 1.

**Question 14-1.** What response is needed to set the tone to its *lowest* value and the volume to *maximum*?

**Question 14-2.** Type:

```
1,0
0,100
```

Explain the result.

**Step 3.** Type:

2,101 Write 101 to channel B fine-tune tone register.
7,60 Enable both channels A and B.

8,10 Set the volume for channel A to 10.
9,10 Set the volume for channel B to 10.

**Question 14-3.** Explain the effect produced in step 3.

**Step 4.** Reset the PSG by touching pin 23 to ground. With the test program still running, type:

7,7 Enable noise on all three channels.
8,10 Set the volume of all channels to 10.
9,10
10,10

You should hear a *rushing* sound. Try writing numbers to register 6 between 0 and 31. This will change the noise generator frequency.

**Step 5.** With the pattern of step 4 established, try the following:

6,15 Set the noise frequency to midvalue.
8,16 Enable the envelope generator mode for all three channels.
9,16
10,16
12,16 Set a short decay time.
13,0 Set envelope decay for one cycle only.

After typing 13,0 you should hear a *single gunshot*. Repeatedly pushing the ENTER key will repeat the effect.

**Step 6.** Change the registers by typing:

6,0 Set the noise frequency to its lowest value.
12,56 Set a longer decay time.
13,0 Set the envelope decay for one cycle only.

You should hear the sound of an *explosion*.

**Question 14-4.** With a small change the PSG will now produce the sound of a ringing bell. What changes are needed?

**Step 7.** Modify the answer to Question 14-4 by typing 13,8 in place of 13,0. The bell should now ring on its own (refer to Fig. 14-10). Changing the data written to register 12 will change the delay time.

**Step 8.** Write a program to simulate a *whistling bomb* sound effect using the explosion produced in step 6. A solution is provided at the end of this experiment.

**Step 9.** The following program converts the TRS-80 into an *electronic organ*. Keys A through K correspond to one consecutive octave of notes with $C^{\#}$ at W, $D^{\#}$ at E, $F^{\#}$ at T, $G^{\#}$ at Y and $A^{\#}$ at U. The computer will save your notes (and mistakes!) and display the current number of notes in memory on the screen. The offset allows a vibrato effect.

```
  5 CLEAR 500
 10 CLS:  DIM D(200), N$(500)                :REM STORAGE FOR 500 NOTES
 20 PRINT TAB(15) ;"TRS-80 AY-3-8910
    ELECTRONIC ORGAN"
 30 PRINT
 35 FOR J=0 TO 127:  D(J)=0:  NEXT J
 37 REM D ARRAY HOLDS TONE CODES
    FOR A-K ON KEYBOARD
 40 READ N1,N2                               :REM GET ASCII KEY AND
                                              TONE CODE
 50 IF N1=999 THEN 80                        :REM END OF DATA
 60 D(N1)=N2                                 :REM ASSIGN TONE CODE TO
                                              ARRAY
 70 GOTO 40
 80 OUT 236,16                               :REM MODEL III ONLY
 90 OUT 127,128                              :REM INIT PPI
100 R=7:D=63:GOSUB 1000                      :REM DISABLE ALL CHANNELS
110 INPUT "ENTER AN OFFSET FROM
    THE MAIN CHANNEL (OR 0):  ";BOFF
130 PRINT
140 INPUT "DO YOU WANT FIXED OR
    ENVELOPE CONTROL (F/E) ";V$
145 PRINT
150 IF LEFT$(V$,1)="F" THEN 220
160 D=16:R=8:GOSUB 1000                      :REM ENABLE ENVELOPE MODE
170 R=9:GOSUB 1000
190 INPUT "ENTER THE DECAY
    VARIABLE (0-255):  ";DELAY
200 R=12:D=DELAY:GOSUB 1000
210 GOTO 260
220 INPUT "WHAT VOLUME LEVEL
    (0-15):  ";V
230 R=8:D=V:GOSUB 1000
240 R=9:GOSUB 1000
245 R=0:D=0:GOSUB 1000                       :REM BE SURE TONE IS OFF
247 R=2:  GOSUB 1000
260 PRINT
290 PRINT "THE COMPUTER WILL SAVE
    YOUR NOTES. Q PLAYS BACK
    NOTES."
300 R=7:D=60:GOSUB 1000                      :REM ENABLE TONE ON
                                              A AND B
```

```
 500  N$=INKEY$
 505  IF INKEY$="" THEN 500
 506  IF N$="Q" THEN 3000
 507  N$(I)=N$:I=I+1:PRINT@1015,I;
 510  N=ASC(N$)
 540  R=0:D=D(N):GOSUB 1000                  :REM PLAY TONE ON
                                              CHANNEL A
 560  R=2:D=D(N)-BOFF:GOSUB 1000             :REM PLAY B WITH ANY
                                              OFFSET
 600  R=13:D=0:GOSUB 1000                    :REM ENVELOPE DECAY
                                              1 CYCLE ONLY
 610  GOSUB 500
1000  IF D>255 OR D<0 THEN D=0
1010  OUT 124,R:OUT 126,3:OUT 126,0:
      OUT 124,D:OUT 126,2:OUT 126,0
1020  RETURN
3000  PRINT
3002  INPUT "ENTER THE TEMPO
      (0-500): ";T
3003  FOR J=0 TO I-1
3005  N=ASC(N$(J))
3020  R=0:D=D(N):GOSUB 1000                  :REM PLAY NOTE ON A
3040  R=2:D=D(N)-BOFF:GOSUB 1000             :REM PLAY THE OFFSET NOTE
3070  R=13:D=0:GOSUB 1000                    :REM ENVELOPE DECAY
                                              1 CYCLE ONLY
3075  FOR K=1 TO T:NEXT K                    :REM TEMPO
3080  NEXT J
3090  FOR K=1 TO 1000:NEXT K                 :REM ALLOW LAST NOTE TO
                                              DIE
3100  CLS: GOTO 80
4000  DATA 65,214,87,202,83, 190,69,180,68,
      170,70,160,84,151
4010  DATA 71,143,89,135,72,127,85,120,74,
      113,75,107,999,999
```

**Step 10.** Reread the objectives as stated at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

## SOLUTIONS TO QUESTIONS

14-1. 1,15 Lowest tone frequency.
8,15 Highest volume.

14-2. The coarse-tune register for channel A is set to 0 and the fine-tune register is set to 100. The new tone produced is of a much higher frequency.

**14-3.** Because the channel A tone (register 0) = 100 but channel B tone = 101 (register 2), and both channels are enabled, a resonance effect is heard.

**14-4.** 0,100 Set all three channels for a high-pitched bell tone.
2,100
4,100
7,56 Enable tone only for channels A to C.
13,0 One decay cycle only.

STEP 8 (solution)

```
 10 CLS
 20 OUT 236,16                        :REM MODEL III ONLY
 30 OUT 127,128                       :REM INIT PPI
 40 R=7:D=62:GOSUB 500                :REM ENABLE TONE ON
                                       CHANNEL A ONLY
 50 R=8:D=15:GOSUB 500                :REM MAXIMUM VOLUME
 60 FOR J=48 TO 192
 70 R=0:D=J:GOSUB 500                 :REM SWEEP EFFECT
 80 NEXT J
 90 R=6:D=15:GOSUB 500                :REM SET NOISE TO
                                       MIDVALUE
100 R=7:D=7:GOSUB 500                 :REM ENABLE NOISE ON
                                       ALL CHANNELS
105 R=8:D=16:GOSUB 500                :REM ENVELOPE MODE
110 R=9:GOSUB 500
120 R=10:GOSUB 500
130 R=12:D=56:GOSUB 500               :REM DELAY TIME FOR
                                       EXPLOSION
140 R=13:D=0:GOSUB 500                :REM DECAY ONE CYCLE
                                       ONLY
150 INPUT A$                          :REM DO IT AGAIN
160 GOTO 40
500 OUT 124,R:OUT 126,3:OUT 126,0
510 OUT 124,D:OUT 126,2:OUT 126,0
520 RETURN
```

# Appendices

# APPENDIX A

## DESCRIPTION OF THE MODEL I AND MODEL III EXPANSION CONNECTORS

The model I TRS-80 was designed to be expanded *external* to the main keyboard control unit via the *expansion interface*. Specifically, the expansion interface provides capabilities for increased memory capacity (to 48K), a disk drive controller, parallel and serial printer interfaces, and a real-time clock. To support these external devices the keyboard control unit must provide nearly *all* the Z-80 CPU control signals at its expansion connector. This includes the *data bus*, full 16-bit *address bus*, and *control signals*.

When the Model III computer was designed, the design philosophy was changed to one of full *internal* expansion. All the interfaces listed above, including room for two mini-disk drive units, are provided within the Model III cabinet.

For this reason, the expansion bus on the Model III (referred to as the *IO bus* by Radio Shack) is *not* as versatile as that on the Model I computer. The most significant limitation is that the *high-order address bus* signals (A8 to A15) and the control bus signals $\overline{RD}$ and $\overline{WR}$ are *not* brought out to this connector. This limits all I/O interfacing to the *I/O-mapped I/O* variety. Apparently, Radio Shack felt that with the Model III computer fully expanded to 48K, there would be no room for memory-mapped I/O devices and therefore deleted these address and control lines from the expansion connector.

### CONNECTOR PINOUTS

Figures A-1 and A-2 indicate the specific signals on each pin of the expansion connector for both computer models.

Another significant difference between the two computers is that the

| P/N | SIGNAL NAME | DESCRIPTION |
|---|---|---|
| 1 | RAS* | Row Address Strobe Output for 16-Pin Dynamic Rams |
| 2 | SYSRES* | System Reset Output, Low During Power Up Initialize or Reset Depressed |
| 3 | CAS* | Column Address Strobe Output for 16-Pin Dynamic Rams |
| 4 | A1Ø | Address Output |
| 5 | A12 | Address Output |
| 6 | A13 | Address Output |
| 7 | A15 | Address Output |
| 8 | GND | Signal Ground |
| 9 | A11 | Address Output |
| 10 | A14 | Address Output |
| 11 | A8 | Address Output |
| 12 | OUT* | Peripheral Write Strobe Output |
| 13 | WR* | Memory Write Strobe Output |
| 14 | INTAK* | Interrupt Acknowledge Output |
| 15 | RD* | Memory Read Strobe Output |
| 16 | MUX | Multiplexor Control Output for 16-Pin Dynamic Rams |
| 17 | A9 | Address Output |
| 18 | D4 | Bidirectional Data Bus |
| 19 | IN* | Peripheral Read Strobe Output |
| 20 | D7 | Bidirectional Data Bus |
| 21 | INT* | Interrupt Input (Maskable) |
| 22 | D1 | Bidirectional Data Bus |
| 23 | TEST* | A Logic "Ø" on TEST* Input Tri-States AØ-A15, DØ-D7, WR*, RD*, IN*, OUT*, RAS*, CAS*, MUX* |
| 24 | D6 | Bidirectional Data Bus |
| 25 | AØ | Address Output |
| 26 | D3 | Bidirectional Data Bus |
| 27 | A1 | Address Output |
| 28 | D5 | Bidirectional Data Bus |
| 29 | GND | Signal Ground |
| 30 | DØ | Bidirectional Data Bus |
| 31 | A4 | Address Bus |
| 32 | D2 | Bidirectional Data Bus |
| 33 | WAIT* | Processor Wait Input, to Allow for Slow Memory |
| 34 | A3 | Address Output |
| 35 | A5 | Address Output |
| 36 | A7 | Address Output |
| 37 | GND | Signal Ground |
| 38 | A6 | Address Output |
| 39 | ~~+5V~~ GND | ~~5 Volt Output (Limited Current)~~ Signal Ground |
| 40 | A2 | Address Output |
| NOTE: *means Negative (Logical "Ø") True Input or Output | | |

Figure A-1 Signal descriptions for the Model I TRS-80 expansion bus.

Model III expansion bus is not only fully buffered through tri-state gates from the "outside world," but it can also be *disabled* under software control. In fact, when powering up the Model III, its expansion connector will be "*dead*" and not respond to any INP or OUT commands. Only after bit *4* of port $236_{10}$ is made a logic 1 (OUT **236,16**) will the external I/O bus be enabled.

Unfortunately, BASIC tends to write into this port from time to time while your program is running, thereby disabling the expansion port. For

| P/N | Signal name | Description |
|---|---|---|
| 1 | D0 | Bidirectional data bus |
| 3 | D1 | Bidirectional data bus |
| 5 | D2 | Bidirectional data bus |
| 7 | D3 | Bidirectional data bus |
| 9 | D4 | Bidirectional data bus |
| 11 | D5 | Bidirectional data bus |
| 13 | D6 | Bidirectional data bus |
| 15 | D7 | Bidirectional data bus |
| 17 | A0 | Address output |
| 19 | A1 | Address output |
| 21 | A2 | Address output |
| 23 | A3 | Address output |
| 25 | A4 | Address output |
| 27 | A5 | Address output |
| 29 | A6 | Address output |
| 31 | A7 | Address output |
| 33 | $\overline{IN}$ | Peripheral read strobe output |
| 35 | $\overline{OUT}$ | Peripheral write strobe output |
| 37 | $\overline{Reset}$ | System reset output |
| 39 | $\overline{IOINT}$ | Interrupt input (maskable) |
| 41 | $\overline{IOWAIT}$ | Processor wait input |
| 43 | $\overline{EXTIOSEL}$ | Data bus enable input for I/O read |
| 45 | NC | No connection |
| 47 | $\overline{M1}$ | M1 machine cycle (op code fetch) |
| 49 | $\overline{IOREQ}$ | Low output indicates valid I/O address on A0-A7 |
| 2-50 | GND | Signal ground |

Figure A-2 Signal descriptions for the Model III TRS-80 expansion bus.

example, the **CLS** command will *disable* external I/O. For this reason, if your external hardware does not seem to be functioning, you may need to insert an extra **OUT 236,16** in a program loop.

One other consideration must be made when interfacing to the Model III. Because its data bus is fully buffered, the *direction* of data on the bus (in or out) must be controlled by external hardware. Pin 43, $\overline{EXT\ I/O\ SEL}$, must be *low* for an **INP** command but *high* for an **OUT** command.

By tracing pin 43 on the schematic diagram in Fig. A-3 you can see that this line is internally pulled high and therefore the data bus is normally set up to *output* data.

Interfacing to this bus is not as difficult as it might seem. Ordinarily, the $\overline{IN}$ control line can simply be connected to $\overline{EXT\ I/O\ SEL}$, thereby automatically enabling the data bus in the proper direction when the **INP** command occurs. Using this technique, each input port must have its own unique port address and tri-state buffers to avoid two devices trying to drive the bus simultaneously.

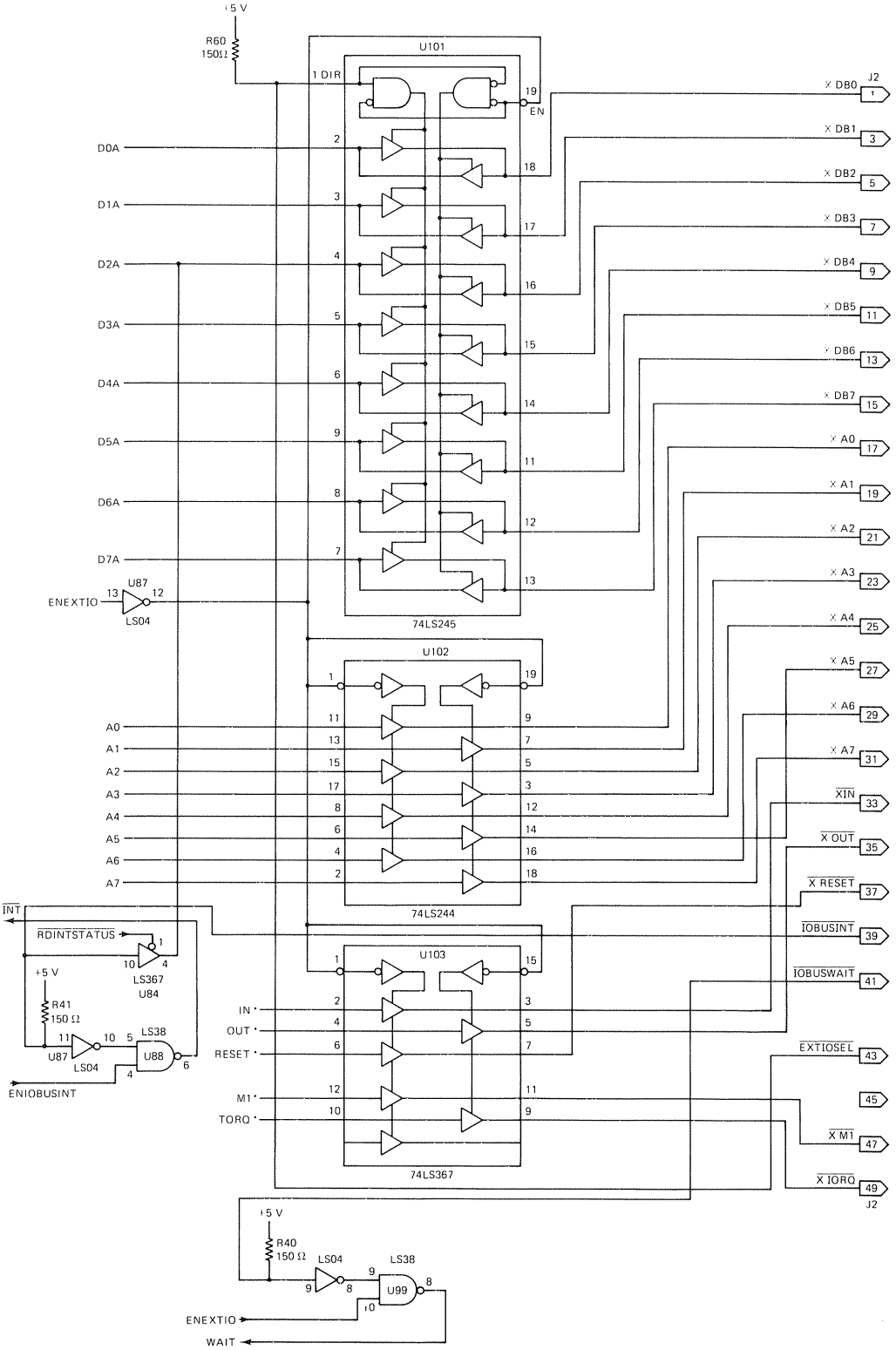



Figure A-3 Schematic diagram of the Model III expansion port. Notice that $\overline{\text{EXT I/O SEL}}$ (pin 43) must be brought low to input data.

## CONNECTING THE INTERFACE CABLE TO THE COMPUTER

The connectors required for interfacing to the two computers are both of the card edge type, but the Model I requires a *40*-pin connector, while the Model III needs a *50*-pin connector. Figures A-4 and A-5 illustrate the location of the expansion connector for both computers. The location of pin 1 is also pointed out in each sketch.

For the experiments in this book, access to the expansion bus is obtained by using a ribbon cable with *card edge* connector on one end and *socket* connector on the other. The following steps should be followed when connecting this cable to the computer.

1. Turn all computer power off.
2. Locate the card edge connector end of the cable. Notice that there is an arrow or other marking identifying one end of the cable. Slide the connector over the TRS-80 card edge noting to which pin number this mark is aligned.
3. Inspect the socket connector end of the cable and again locate the arrow mark. This will identify the pin number you aligned the card edge to. Note that the pins in one row are all even numbered, whereas those on the opposite side are odd numbered.

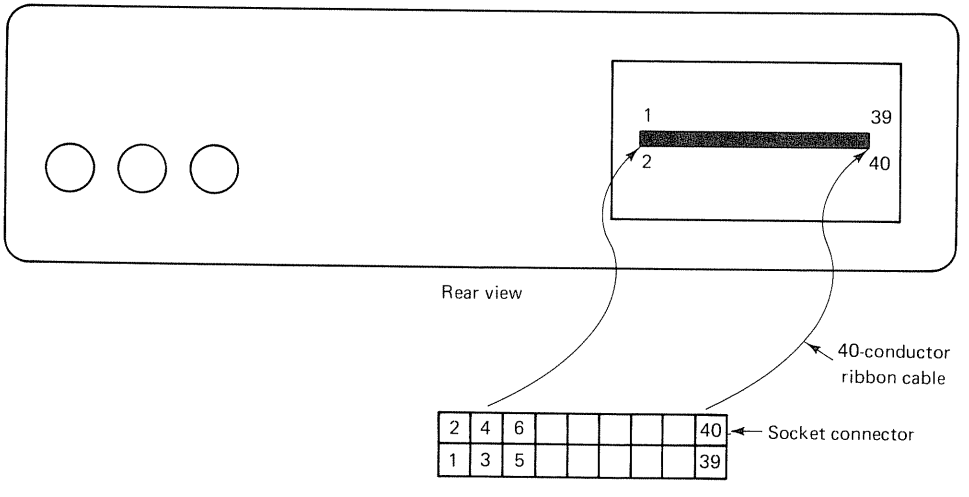
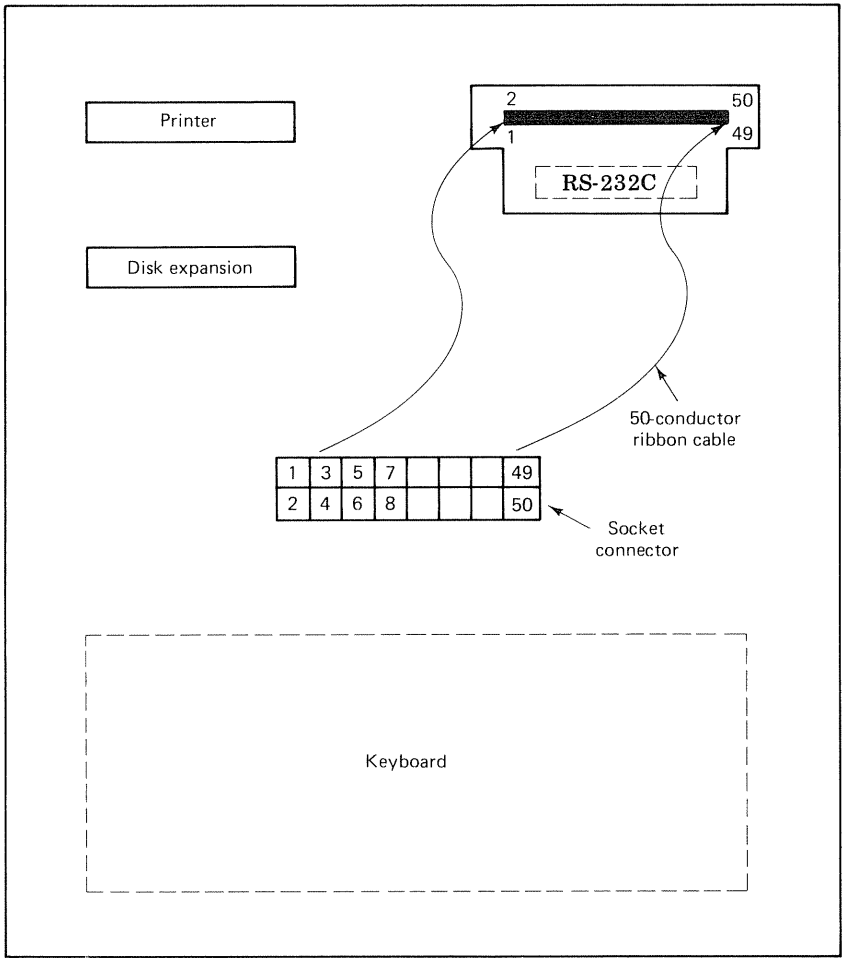



Figure A-4 A 40-conductor ribbon cable is used to access the expansion port of the Model I TRS-80.




Bottom view

Figure A-5 A 50-conductor ribbon cable is used to access the expansion port of the Model III TRS-80.

*Note to Model I users with the expansion interface.* Although there is a "duplicate" expansion connector on the expansion interface unit, the $\overline{RD}$ amd $\overline{WR}$ lines are not *active* at this connector. When doing the *memory-mapped* I/O experiments in this book, you should temporarily *disconnect* the expansion interface and connect directly to the keyboard unit.

# APPENDIX B

## PARTS LIST

Tables B-1 to B-4 list all the parts required to perform the 14 experiments in this book. The following information is supplied:

1. Part number
2. Description of the part
3. Quantity needed to do any one experiment
4. Supplier code
   - RS Radio Shack
   - J Jameco Electronics, 1355 Shoreway Road, Belmont, CA 94002
   - D Digi-Key Corp., P.O. Box 677, Thief River Falls, MN 56701
   - P Priority One Electronics, 9161B Deering Avenue, Chatsworth, CA 91311

Most of the parts are noncritical and slight variations should present no problems in the experiments (for example, substituting a 330-Ω for a 300-Ω resistor). Many other suppliers also exist and you should consult any of the popular hobbyist magazines.

TABLE B-1 PARTS LIST: INTEGRATED CIRCUITS

| Part | Description | Quantity | Supplier | Number per experiment | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 7400 | Quad NAND gates | 1 | RS J D P | | | | | | | 1 | | 1 | | | | | 1 |
| 7404 | Hex inverters | 2 | RS J D P | | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 7430 | 8-input NAND gate | 1 | J D P | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 7432 | Quad OR gates | 1 | RS J D P | | 1 | 1 | 1 | | | | | | | | | | |
| 7476 | Dual JK flip-flop | 1 | RS J D P | 1 | 1 | | | 1 | 1 | | | | | | | | |
| 7485 | 4-bit comparator | 1 | J D P | | | 1 | | | | | | | | | | | |
| 74100 | 8-bit latch | 1 | J | | | 1 | | | | | | | | | 1 | | |
| 74121 | One-shot | 1 | J P | | | | | | | | | | | | | | 1 |
| 74LS244 | Tri-state buffers | 1 | J D | | | | 1 | | | | | | | | | | |
| 2114 | 1K X 4 RAM | 2 | RS J D P | | | | | | | 2 | | | | | | | |
| 8255 | PPI | 1 | J D P | | | | | 1 | 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| MC1408 | 8-bit DAC | 1 | J P | | | | | | | | | | 1 | 1 | | | |
| TIL111 | Opto-coupler | 1 | RS | | | | | | | | | 1 | | | | | |
| DL707 | Display | 1 | RS J P | | | | | | | | | | | | 1 | | |
| AY-5-1013 | UART | 1 | J P | | | | | | | | | | | | | 1 | |
| AY-3-8910 | PSG | 1 | see Exp. 14 | | | | | | | | | | | | | | 1 |
| 4013 | Dual flip-flop | 1 | RS J D P | | | | | | | | | | | | | | 1 |
| 4069 | Hex inverters | 1 | J D P | | | | | | | | | | | | | | 1 |
| LM334 | Temperature sensor | 1 | RS J D | | | | | | | | 1 | | | 1 | | | |
| LM339 | Quad comparators | 1 | RS J D P | | | | | | | | 1 | | | 1 | | | |
| LM386 | Audio amplifier | 1 | RS J D | | | | | | | | | 1 | | | | | 1 |
| LM555 | Timer | 1 | RS J D P | | | | | | | | | | | | | 1 | |
| LM567 | Tone decoder | 1 | RS J D | | | | | | | | | 1 | | | | | |
| LM1458 | Dual op-amp | 1 | RS J D P | | | | | | | | | | 1 | 1 | | | |

TABLE B-2 PARTS LIST: RESISTORS

| Part | Description | Quantity | Supplier | Number per experiment | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 68 Ω | Resistor | 1 | RS J D P | | | | | | | | | | | | 1 | | |
| 100 Ω | Resistor | 1 | RS J D P | | | | | | | | | 1 | | | | | |
| 180 Ω | Resistor | 8 | RS J D P | 1 | 1 | 8 | | 6 | 6 | | | | 2 | 1 | 2 | 1 | |
| 220 Ω | Resistor | 1 | RS J D P | | | | | | | | 1 | 1 | | 1 | | | |
| 270 Ω | Resistor | 1 | RS J D P | | | | | | | | | | 1 | 1 | | | |
| 300 Ω | Resistor | 1 | RS J P | | | | | | | | | | | | | | 1 |
| 330 Ω | Resistor | 1 | RS J D P | | | | | | | | | 1 | | | | | |
| 470 Ω | Resistor | 1 | RS J D P | | | | | | | | | | | | | | 1 |
| 1 kΩ | Resistor | 10 | RS J D P | | | | 8 | 2 | 2 | | 4 | 2 | 2 | 3 | 10 | 1 | 1 |
| 4.7 kΩ | Resistor | 2 | RS J D P | | | | | | | | | | | | | | 2 |
| 10 kΩ | Resistor | 2 | RS J D P | | | | | | | | 2 | | 1 | 2 | | | |
| 10 MΩ | Resistor | 1 | RS J D P | | | | | | | | | | | | | | 1 |
| 10 kΩ | Potentiometer | 2 | RS J D P | | | | | | | | 2 | 2 | 1 | 2 | | | |
| 50 kΩ | Potentiometer | 1 | RS J D P | | | | | | | | | | | | | 1 | |
| | Photo cell | 1 | RS | | | | | | | | 1 | | | | | | |

TABLE B-3 PARTS LIST: CAPACITORS

| Part | Description | Quantity | Supplier | Number per experiment | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 20 pF | Capacitor | 1 | J D | | | | | | | | | | | | | | 1 |
| 100 pF | Capacitor | 1 | RS J D | | | | | | | | | | 1 | 1 | | | 1 |
| 300 pF | Capacitor | 1 | J D | | | | | | | | | | | | | | 1 |
| 0.001 μF | Capacitor | 1 | RS J D | | | | | | | | | | | | | | 1 |
| 0.0047 μF | Capacitor | 1 | RS J D | | | | | | | | | 1 | | | | | |
| 0.005 μF | Capacitor | 1 | RS J D | | | | | | | | | 1 | | | | | |
| 0.01 μF | Capacitor | 1 | RS J D | | | | | | | | | | | | | 1 | |
| 0.033 μF | Capacitor | 1 | J D | | | | | | | | | 1 | | | | | |
| 0.05 μF | Capacitor | 1 | RS J D | | | | | | | | | 1 | | | | | |
| 0.1 μF | Capacitor | 1 | RS J D | | | | | | | | | 1 | | | | | 1 |
| 10 μF | Capacitor | 1 | RS J D | | | | | | | | | 1 | | | | | 2 |
| 100 μF | Capacitor | 1 | RS J D | | | | | | | | | 1 | | | | 1 | 1 |

TABLE B-4 PARTS LIST: MISCELLANEOUS

| Part | Description | Quantity | Supplier | Number per experiment | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| LED | Red | 8 | RS J D P | 1 | 1 | 8 | | 2 | 2 | | | 1 | 1 | 1 | 1 | 1 | |
| LED | Green | 2 | RS J P | | | | | 2 | 2 | | | | | | 1 | | |
| LED | Yellow | 2 | RS J P | | | | | 2 | 2 | | | | | | | | |
| Switch | PBNO | 2 | RS J D | | | | | 2 | 2 | | | | | | 2 | | |
| Switch | Magnetic | 1 | RS | | | | | | | | 1 | 1 | | | | | |
| Switch | 8-position DIP | 1 | RS J D | | | 1 | 1 | | | | | | | | 1 | | |
| Buzzer | Piezoelectric | 1 | RS J D | | | | | | | | | 1 | | | | | |
| Transistor | *npn* | 1 | RS J D P | | | | | | | | | 1 | | | | | |
| Transducer | 40 kHz | 2 | see Exp. 9 | | | | | | | | | 2 | | | | | |
| Relay | SPDT DIP | 1 | RS J | | | | | | | | | 1 | | | | | |
| Crystal | 3.579545 MHz | 1 | RS J D P | | | | | | | | | | | | | | 1 |
| Diode | General purpose | 1 | RS J D P | | | | | | | | | 1 | | | | | |
| | Loud speaker | 1 | RS J | | | | | | | | | | | | | | 1 |

# APPENDIX C

## BINARY AND DECIMAL NUMBERS

All digital computers work with electrical circuits that can be considered *ON* or *OFF*. Usually, this is referred to as a *1* or a *0*, or a *high* and a *low*. Because of this *ON-OFF* nature, the *binary* number system is used to represent data and addresses within the computer.

Just as decimal is a base *ten* number system, binary is a base *two* number system. Each binary digit or *bit* has a value or weight that is a *factor of 2* higher than the digit to its right. The value of any digit is found as the base (2) raised to the power corresponding to its position from the right. The first position is considered zero. For example,

$$1101 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 8 + 4 + 0 + 1 = 13_{10}$$

Binary numbers are converted to their decimal equivalents by simply adding the appropriate binary weights. Table C-1 lists the binary value for each of the first 16-bit positions. Notice that when moving from right to left, each bit is a factor of 2 higher than its rightmost neighbor.

The TRS-80 uses *8*-bit words and some typical examples are:

$$00001011 = 8 + 2 + 1 = 11_{10}$$
$$00100110 = 32 + 4 + 2 = 38_{10}$$
$$11000111 = 128 + 64 + 4 + 2 + 1 = 199_{10}$$
$$11111111 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255_{10}$$

TABLE C-1 POWERS OF 2 FOR THE FIRST 16 BIT POSITIONS

| Bit position ⟶ | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Binary weight | 32,768 | 16,384 | 8192 | 4096 | 2048 | 1024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

When converting from decimal to binary, the highest remaining power of 2 is repeatedly *subtracted* from the decimal number. For example,

| | | |
|---|---|---|
| | $25_{10}$ | |
| $(2^4)$ | $-\ \underline{16}$ | (highest power of 2 in 25) |
| | 9 | |
| $(2^3)$ | $-\ \underline{8}$ | (highest power of 2 in 9) |
| | 1 | |
| $(2^0)$ | $-\ \underline{1}$ | (highest power of 2 in 1) |
| | 0 | |

The answer is then the binary number corresponding to $2^4(16) + 2^3(8) + 2^0(1)$ or 11001. Other examples are:

$$58 = 2^5(32) + 2^4(16) + 2^3(8) + 2^1(2) = 111010$$

$$147 = 2^7(128) + 2^4(16) + 2^1(2) + 2^0(1) = 10010011$$

$$250 = 2^7(128) + 2^6(64) + 2^5(32) + 2^4(16) + 2^3(8) + 2^1(2) = 11111010$$

# APPENDIX D

## BASIC LOGIC GATES

There are only *three* basic logic functions used in any digital system. These are the *AND*, *OR*, and *NOT* functions. Schematically these are represented as logic gates and illustrated in Fig. D-1.

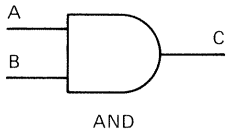
The *AND* gate requires A AND B to be high in order for the output C to be high . The *OR* gate requires A OR B to be high in order for its output C to be high. The *NOT* gate or inverter simply inverts the value of its input: a 1 in yields a 0 out, and a 0 in yields a 1 out.

Although the AND and OR gates in Fig. D-1 have only two inputs (A and B), logic gates with more than two inputs are also available.

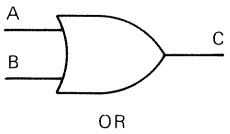
Two common variations of the AND and OR gate are the *NAND* (NOT-AND) and *NOR* (NOT-OR) gates. These are illustrated in Fig. D-2. From their truth tables (and logic symbols) it can readily be seen that these gates simply provide the *inverted* output of their AND and OR gate equivalents. Thus a NAND gate has a 0 output when inputs A AND B are high, while the NOR gate provides a 0 output when inputs A OR B are high.

Although the truth tables for the various logic gates can (and probably should) be memorized, their schematic symbols also suggest a word interpretation. Refer to Fig. D-3a for the AND gate. If we use the convention that a circle represents an inversion or low (0) logic level, then the standard AND gate symbol says in words: When inputs A and B are high, output C will be high. However, by referring to the AND gate truth table in Fig. D-1a, we can also see that any time A OR B is a 0, the output is also a 0. This suggests the second symbol drawn in Fig. D-3a. The word interpretation is: When A OR B is low, the output C is low. Note how the small circles are used and interpreted for this alternate AND gate symbol. Figures D-3b-(d) illustrate the corresponding interpretation for the OR, NAND, and NOR gates and their alternate symbols.

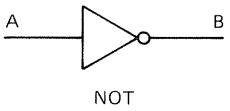
You might wonder why these other logic symbols are used. The answer




| A | B | C |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |




| A | B | C |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

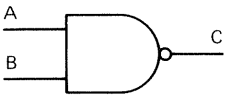



| A | B |
|---|---|
| 0 | 1 |
| 1 | 0 |

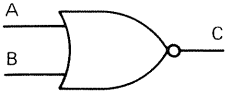
Figure D-1 The three basic logic gates and their truth tables: (a) AND; (b) OR; (c) NOT.

is that often logic signals are *active low* (go to a 0 level when they are present) and we may need to perform any of the various logic functions on these active low signals. Figure D-4 illustrates this case. In this figure we need to determine when two active low signals (note the ⌐⌊⌐ symbol), $\overline{\text{SEL219}}$ AND $\overline{\text{IN}}$, are *both* low and then produce an active high output signal (IN 219).

Although the circuits in parts a and b of Fig. D-4 are equivalent and both use the same two-input NOR gate, Fig. D-4a is much easier to understand in terms of what the circuit is doing than is the equivalent circuit in Fig. D-4b.




| A | B | C |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




| A | B | C |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Figure D-2 Truth tables and logic symbols for the (a) NAND and (b) NOR gates.

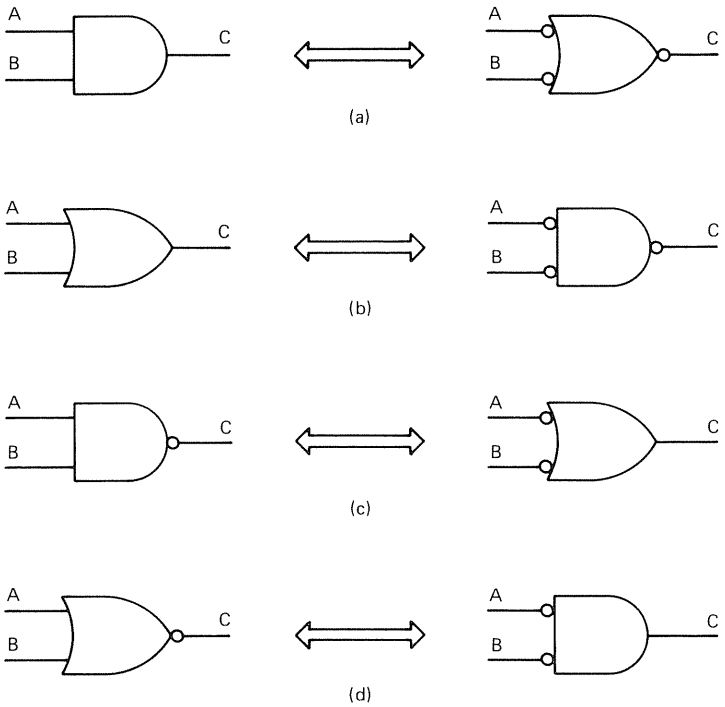



Figure D-3 The primary and alternate (active low) logic symbols are shown for (a) the AND gate, (b) the OR gate, (c) the NAND gate, and (d) the NOR gate.

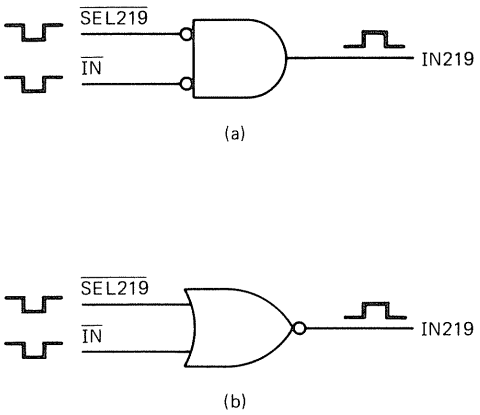



Figure D-4 A NOR gate is used to detect when $\overline{\text{SEL219}}$ AND $\overline{\text{IN}}$ are both low and generate an active high output pulse. Both circuits (a) and (b) are equivalent, but only (a) gives the proper word interpretation.

# APPENDIX E

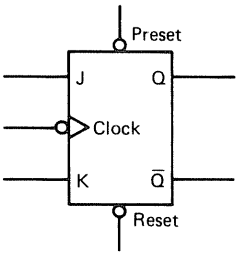
## JK FLIP-FLOP

A flip-flop is a logic element that is used to *store* or remember a logic condition. Typically, a *clock* signal is applied to the flip-flop and at the instant this clock occurs, the flip-flop *latches* the binary signal it is monitoring.

There are several different types of flip-flops, but the most versatile is the *JK* type. The symbol for this flip-flop is illustrated in Fig. E-1. There are two sets of inputs, referred to as the *asynchronous* (PRESET and RESET) and *synchronous* (J and K) inputs.

The flip-flop responds to the asynchronous inputs *immediately*, independent of the clock signal. Studying the first three entries in the truth table, when the PRESET input is low, the Q output is *set* or high. Similarly, when the RESET input is low, the Q output is *reset* or low. Because it is not logical to try to set and reset the flip-flop at the same time, the PRESET and RESET inputs should not both be low at the same instant.

The synchronous inputs, J and K, are used only when the PRESET and RESET inputs are both high. This corresponds to the last four entries in the table. Using J and K, the flip-flop output will change only when the clock




| Asynchronous inputs | | Synchronous inputs | | Output |
|---|---|---|---|---|
| Preset | Reset | J | K | $Q_{t+1}$ |
| 0 | 1 | X | X | 1 |
| 1 | 0 | X | X | 0 |
| 0 | 0 | X | X | Not allowed |
| 1 | 1 | 0 | 0 | No change |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | Toggle |

**Figure E-1 Truth table and logic symbol for the JK flip-flop.**

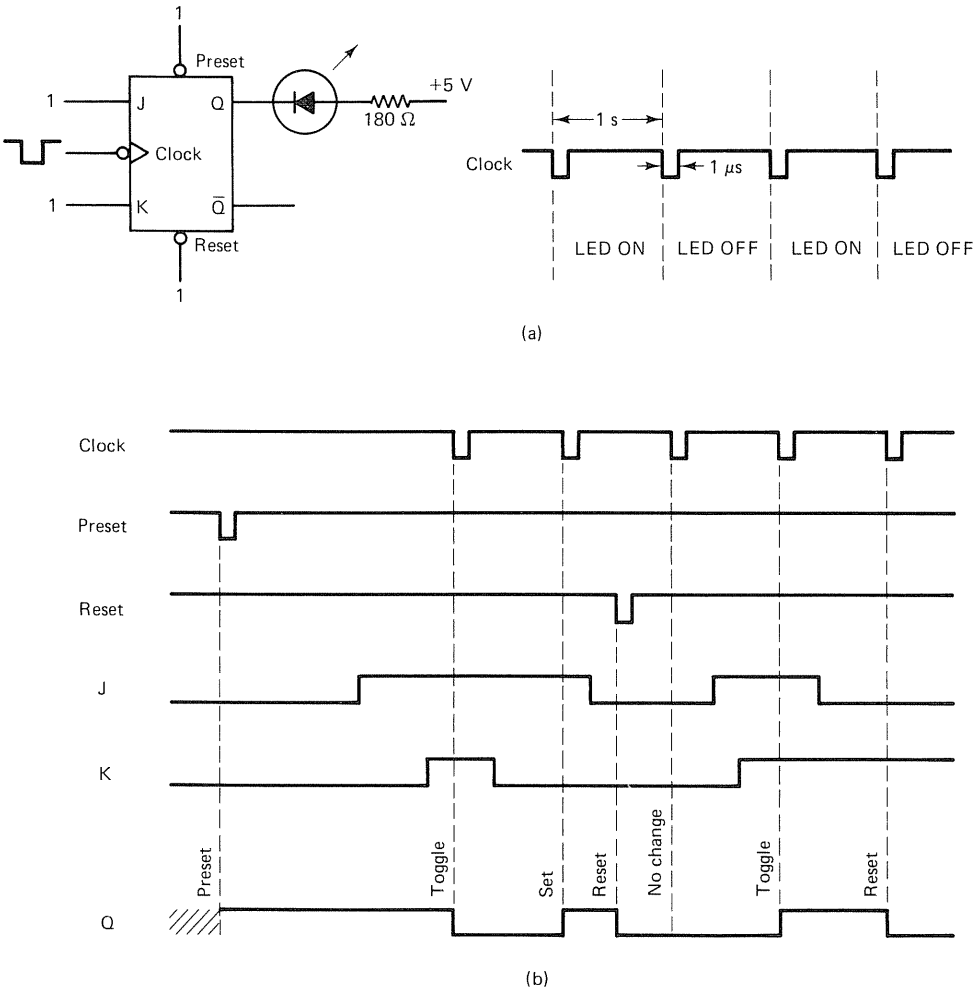



Figure E-2 In (a) the flip-flop is used to detect very short pulses; in (b) it responds to its J and K inputs when the falling edge of the clock pulse occurs.

pulse occurs. Depending on the logic levels of J and K, the Q output may do nothing (no change state), reset (Q = 0), set (Q = 1), or toggle (switch to the opposite state).

In this book the JK flip-flop is often used to catch pulses that are too short to be observed normally (1 to 2 μs long). Figure E-2a illustrates the technique. With J and K high, the flip-flop will toggle each time the pulse arrives. When Q is low, the LED will light. In this way we can observe each time a pulse occurs even though the pulse itself is only 1 μs wide.

Finally, Fig. E-2b illustrates the general case where all the inputs are changing in time and the flip-flop responds accordingly.

# APPENDIX F

## IF THE EXPERIMENT DOESN'T WORK

One of *Murphy's laws* states: If anything can go wrong, it will. As you do the experiments in this book you may decide to rewrite this law to say: The more wires you plug into a logic breadboard, the less chance the circuit has of working! This leads us to Rule 1 when working with digital circuits:

*Rule 1.* Keep calm.

It is surprising how many of yesterday's major problems can be solved the next morning in five minutes with a cool head. When you find yourself getting agitated, take a break. Come back to the circuit later. More often than not, you will fix your problem in a few minutes.

*Rule 2.* Keep it neat.

This has to be the *cardinal* rule for wiring digital circuits. Don't use a 6-inch length of wire to connect two points that really require only $\frac{1}{2}$ inch of wire. If you don't heed this rule, you'll soon have a "rat's nest" of wires that is impossible to trace and debug. Problems in this kind of circuit almost always require *complete rewiring* of the circuit.

*Rule 3.* Understand the circuit function.

It is very hard to troubleshoot something if you don't understand how it words in the first place. If you don't know that two 1's into a NAND gate cause a 0 out, you won't be able to recognize a bad chip if you see one. This rule may require you to sit down and study the circuit for awhile. But that's OK. This also helps you to follow Rule 1.

*Rule 4.* Isolate the problem.

This is the key to all good troubleshooting. Many of the circuits and programs in this book get quite complicated, and when you throw in the computer control function, a number of different problems can occur. A first step is to isolate the problem to *hardware* or *software*. You can do this

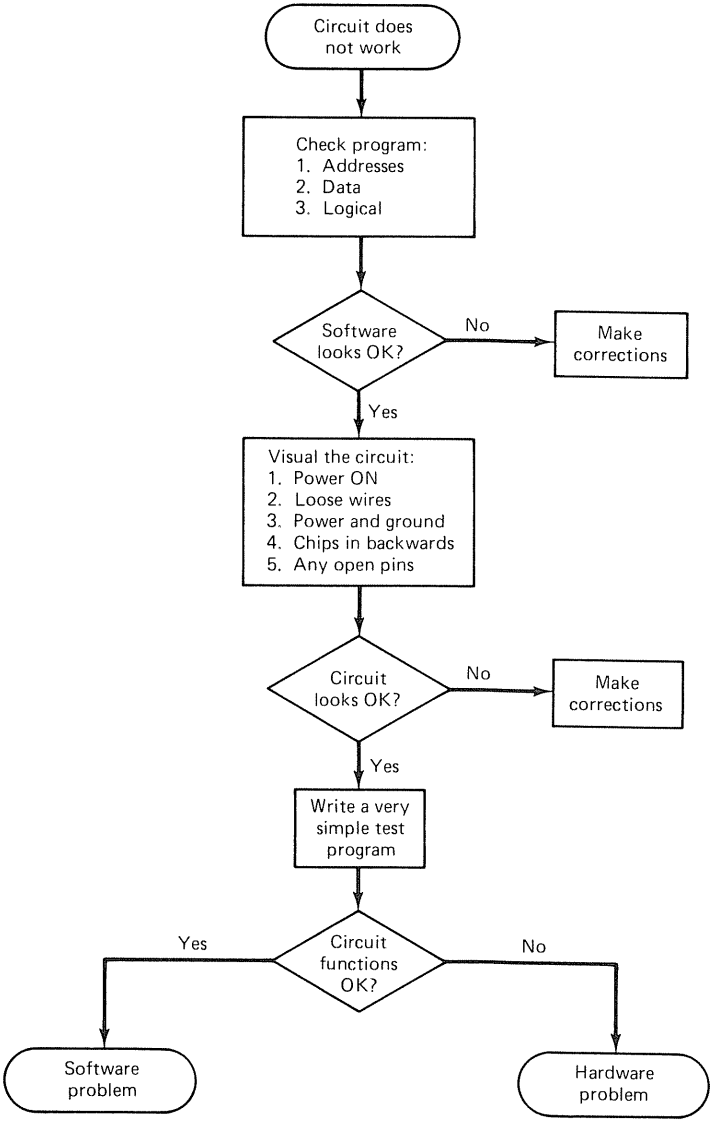



Figure F-1 General flowchart to be used for locating hardware or software problems in the experiments.

by running a *very simple* program that you *know* must work. If the circuit still does not function, it is most likely that you have a *hardware* problem.

Figure F-1 is a flowchart of the logical steps to follow when troubleshooting any of the experiments in this book. After performing obvious checks of the software and hardware for errors, a simple test program should be run. Based on the results of this program, one of two paths is chosen.

### Software Problems

If you think the problem is within the software, double check your program line by line for accuracy. Often it is a good idea to insert *breakpoints* into the program and have the computer output *intermediate* results. What you are trying to do is isolate the problem to a particular portion or command in the program.

### Hardware Problems

Hardware problems can usually be attributed to faulty wiring or bad chips. A *logic probe* is extremely handy for debugging these types of problems. If you do not have a logic probe, a logic-level tester can be built from a single interter and LED as shown in Fig. 10-5.

If possible, set up a *static* (nonchanging) condition and trace through the logic with your probe. For example, many of the experiments utilize the *8255* programmable peripheral interface. You could program all I/O pins to be outputs and then see if each pin can be set high and low by testing with your probe. You can similarly test these pins as inputs by applying +5 V or 0 V to each pin (refer to steps 1 and 2 and Question 10-1 of Experiment 10).

All the experiments require an address decoder and connections to the control bus. With your test program running, you should see pulses on the appropriate control lines, and the address decoder output should pulse when its address appears on the bus.

*Model III users:* Be sure to remember to use **OUT 236,16** to activate your computer's I/O bus and connect $\overline{EXT\ I/O\ SEL}$ (pin 43) to $\overline{IN}$ (pin 33) to configure the data bus properly for input commands (refer to Appendix A for more details).

# INDEX


## A

AC control interface, 111-13, 115 (*See also* relay interface)
Address bus, 17, 26
Address decoder, 27-30
  switch selectable, 43-44
Analog-to-digital converter (ADC):
  block diagram, 132-33
  flash converter, 133-34
  successive approximations, 136-37
  tracking, 134-35
  TRS-80 interface, 137-39
    digital voltmeter, 139-41
    temperature sensor, 141-43
AND command, 50, 52-53
AND gate, 220-21
ASCII, 174-75 (Table 3-3)

## B

Baud rate, 165
Baud rate generator, 168, 171
Binary number system, 217-19
Bus, 16

## C

CMOS, 16
Comparator:
  light sensing interface, 97-100
  LM339, 97-98
  temperature interface, 100-1
Conditional I/O, 148-49
Connector descriptions, 208-9
Control bus, 17, 26

## D

Data bus, 17, 26
Decoder, 27-30
Digital-to-analog converter (DAC):
  full-scale output, 121
  hypothetical 4-bit, 120-21
  MC1408, 122-23
    programmable power supply, 129
    software, 124-26
    TRS-80 interface, 122-24
  ramp generator, 129
  Table 10-1, 127
  triangle wave generator, 129

## F

Flash converter, 133-34
Flip-flop, 223-24

## H

Handshaking, 58, 153
  parallel-printer interface, 154-57

## I

Infrared LED, 111
INP command, 47
Input port:
  I/O-mapped, 48-49
  memory-mapped, 71-73
Interface (TRS-80):
  AC control, 111-13, 115
  DAC, 122-24
  digital voltmeter, 139-41
  magnetic switch, 95-97
  mechanical relay, 108-11
  parallel-printer, 154-57
  photoresistor, 97-100
  PPI (8255), 59-60, 74-76
  programmable power supply, 129
  RAM, 83-87
  ramp generator, 129
  smoke detector, 101-3
  solid-state relay, 111-13
  sound generator, 189-92
  successive approximations ADC, 136-39
  temperature sensor, 100-1, 141-43
  tracking ADC, 134-35
  traffic light controller, 64-65, 77-78
  triangle wave generator, 129
  UART, 168-71
  ultrasonic, 113-15
Interrupt-driven I/O, 150-51
Inverter, 220-21
I/O-mapped I/O, 18
Isolation, 113

## L

Latch, 36-38
  Table 3-1, 38
Logic probe, 9-10

## M

Magnetic switch interface, 95-97
Masking, 50, 52-53
Memory-mapped I/O, 18-21
Model I and Model III compared, 3
Modem, 174-77

## N

NAND gate, 220-21
NOR gate, 220-21

## O

Opto-coupler, 111-12
OR command, 53
OR gate, 220-21
OUT command, 36
Output port:
  I/O-mapped, 36-38
  memory-mapped, 70-71

## P

Partial decoding, 29-30, 73-74
Parts list, 213-16
PEEK command, 72-73
Photoresistor interface, 97-100
POKE command, 70-71
Polled I/O, 148, 150
Port enable pulse, 30-31
Power supply, 5-7
PPI (8255):
  application as traffic light controller, 64-65, 77-78
  block diagram, 58
  general description, 57
  mode 1, 152-54
  modes of operation, 57-58
  software (mode 0), 61-62
  TRS-80 interface:
    I/O mapped, 59-60
    memory-mapped, 74-76
Printer interface (parallel), 154-57

## R

RAM:
  dynamic, 82-83
  static, 82-83
  test program, 87-88
  TRS-80 interface, 83-87

READY flag, 148
Relay interface:
  mechanical, 108-11
  solid-state, 111-13
ROM, 82
RS-232C, 174, 176

## S

Serial interfacing, 163-65
Serial word format, 166-67
Seven-segment display:
  software driven, 158-60
  two-digit output port, 40-41
Smoke detector interface, 101-3
Solderless breadboard, 8
Sound generator:
  AY-3-8910:
    control with BASIC, 192-99
    description, 187-88
    function summary, 198 (Fig. 14-11)
    TRS-80 interface, 189-92
  MK50240N, 185-87
  SN76477, 187-89
  SN76488, 187
  SN76489A, 187, 190

## T

Temperature sensor interface, 100-1
Timing diagrams:
  input read, 19
  memory read, 19
  memory write, 20
  output write, 20
Tone decoder, 115
Tracking ADC, 134-35
Transducer (40KHz interface), 113-15
Transistor relay driver, 109-11
Tri-state gate, 47-48, 51
Troubleshooting, 225-27
TTL, 16

## U

Ultrasonic interface, 113-15
Unconditional I/O, 148-49
Universal asynchronous receiver transmitter (UART):
  description (AY-5-1013), 165-66
  software control, 171-73
  TRS-80 interface, 168-71

## Z

Z-80, 16-17

JOHN E. UFFENBECK

# Hardware Interfacing with the TRS-80

Featuring 14 experiments on interfacing the outside world to the Model I and Model III TRS-80 computers, this book helps you to understand how to use your personal computer to monitor and control electronics interfaces between the computer and your home or work environment. You get a solid "hands-on" feel for the theory; the emphasis is on understanding microcomputer interfacing and control software in BASIC.

**Special features to note:**

- The experiment format helps you to try for yourself the ideas presented.
- All materials needed to perform the experiments are clearly indicated with alternatives and sources.
- The approach is easy to follow, and questions and answers are provided at the end of each experiment.
- All schematic diagrams show part numbers and pin numbers.
- A detailed appendix reviews digital electronics fundamentals, lists all parts needed, and includes a discussion on what steps to take if problems are encountered.

PRENTICE-HALL, INC., Englewood Cliffs, N.J. 07632

ISBN 0-13-383869-2