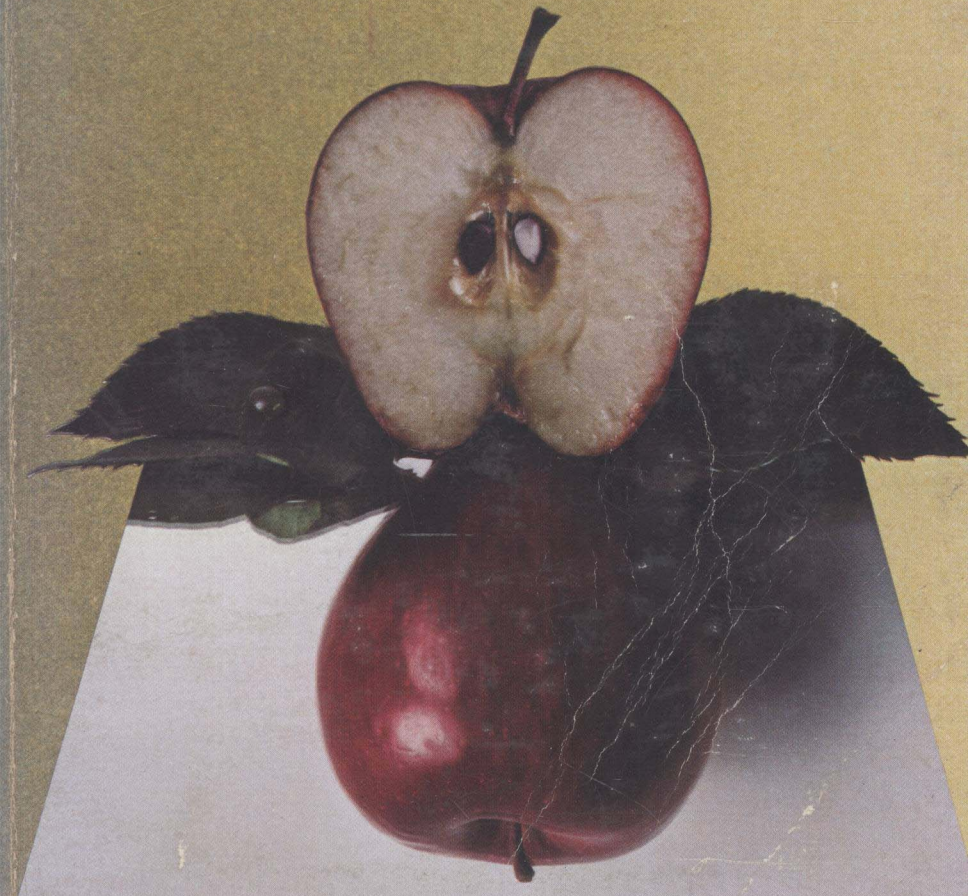
21862

# APPLE® INTERFACING

JONATHAN A. TITUS DAVID G. LARSEN
CHRISTOPHER A. TITUS



BLACKSBURG CONTINUING EDUCATION SERIES™
edited by Larsen, Titus & Titus

# APPLE®
# INTERFACING

by

Jonathan A. Titus, David G. Larsen, and
Christopher A. Titus

Howard W. Sams & Co., Inc.
4300 WEST 62ND ST. INDIANAPOLIS, INDIANA 46268 USA

FIRST EDITION
SECOND PRINTING–1982



International Standard Book Number: 0-672-21862-3
Library of Congress Catalog Card Number: 81-84282

Edited by: *Bob Manville*
Illustrated by: *Jill E. Martin*

*Printed in the United States of America.*

# Preface

The purpose in writing this book is to introduce you to the signals within the Apple®* II computer and to show you how these signals can be used to control external devices under the control of BASIC-language programs. A general-purpose computer interface breadboard has been developed to speed your circuit design and testing so that you can easily perform the many interesting experiments that are included in the book. By using a design system such as the one described in this book, you will spend your time concentrating on the principles involved, rather than troubleshooting your circuits. However, you will have the opportunity to build and test many digital circuits, as well as circuits that use digital-to-analog and analog-to-digital converters.

We have chosen to use the Apple II computer with 16K of read/write memory, and the Applesoft™† BASIC interpreter program. This software provides a great deal of flexibility and it is worth having it available when you are using external interface circuits. The Applesoft BASIC interpreter has two general-purpose commands that can be used to transfer information to and from the computer. These instructions are easily mastered, without requiring a detailed understanding of the 6502 microprocessor integrated circuit (IC) that is used as the "heart" of the Apple.

First we will introduce you to the control signals that are available from the Apple computer for interfacing, and we will show you how they are used. Some of the signals will not be described, since they are generally not used in interface circuits, and are meant to be used by special interface devices that are manufactured commercially.

Our next step is to show you how the Apple can identify or address external devices through the use of two general-purpose instructions, PEEK and POKE. These commands are central to the control of external devices; we spend some time covering their operation and the use of a variety of circuits that can be used to identify specific input/output, or I/O devices. You will also see how the Apple can transfer information to and from external devices over the bidirectional data

---

*Apple and Apple II are registered trademarks of Apple Computer, Inc.
†Applesoft is a trademark of Apple Computer, Inc.

bus; the basic circuits used for *input ports* and *output ports* are described in detail. Real circuits are provided, so that you can quickly use the many examples in designing your own interface devices.

You will also see the power of BASIC-language programs—as the data is processed within the computer to provide meaningful results. Simple control programs are provided to show you how BASIC-language programs and I/O devices can interact. You will be able to write simple control and data processing programs to go along with your I/O ports and devices.

Since the computer is not always synchronized to external devices, there must be some interaction between the computer and the various I/O devices so that each knows when the other is ready for some appropriate action. This leads us to the topic of flags—those signals that are used by the computer and by external I/O devices to allow information to be transferred in an orderly fashion. Since flags are important, we spend some time on them and on the corresponding circuits that are actually used in external devices. Software is covered too, since the flag circuits are useless unless they can be sensed by a control program.

We have assumed that you have a fairly good understanding of the commands in Applesoft BASIC. If you are just getting started with the Apple computer, we hope that you will take some time to review the simple commands, such as FOR, GOTO, IF . . . THEN, PRINT, and INPUT. Other commands will be introduced in the text and experiments, and we will provide the details of their operation. At the end of this book, the use of these and other commands should be second-nature.

In Chapter 6, we have provided 16 detailed, step-by-step experiments that you can perform to reinforce the many interfacing principles that have been developed in the text. You will also see the power of BASIC-language programs for interface control and for actually processing the information that is involved in transfers to and from I/O devices. We have made an effort to cover a broad spectrum of interesting interface applications. Throughout the experiments, you will see that the same basic principles apply to all of the interface circuits, from the simplest to the most complex.

We realize that it is difficult to write a book like this for an audience that has a wide range of backgrounds, from the beginner to the advanced user. Thus, we have chosen to start at some middle point. We have chosen to skip basic binary numbering, decimal-to-binary conversions, basic digital electronics, and breadboarding. These topics are covered in detail in other books, and the reader who is in the middle of our assumed spectrum of readers probably has a good understanding of these topics. In some places, a paragraph or two of review material have been provided, just to serve as a refresher. We

make no attempt to provide much detail here, simply enough to get you started.

We have assumed some familiarity with SN7400-family digital integrated circuits, or chips, such as the SN7402 quad NOR gate and the SN7475 quad latch chip. Other complex chips will be introduced and explained in sufficient detail so that you can use them as shown in the text or experiments. If you wish to use these devices in other applications, we suggest that you obtain the necessary data sheets from the manufacturers. The data sheets will provide the necessary information for a wide variety of uses, and they will also reflect any basic changes or modifications that may have been made to an "updated" device, or one that has been "enhanced" with some special feature.

The Apple II computer has eight general-purpose 50-conductor interface connectors in its case. The basic bus signals used in the experiments are derived from the signals at these connectors, so if you decide to design and build some of your own interface circuits that will be plugged into one of these "slots," you will find the same signals are readily available at the edge connectors. However, there are also some special-purpose signals that are generated by the Apple to make the interfacing task somewhat easier. These signals and their uses are described in detail in Chapter 7. Since the signals are not general purpose, but are specific to the Apple, and in many cases, specific to a particular connector, they are described last. To show you how these signals are used, a simple asynchronous-serial communication interface circuit is described, and software to control it is listed. This type of interface can be used to communicate with other computers, serial printers, modems, and other interface devices that use the asynchronous-serial data format.

We have not described assembly-language programming, since this is a specialized topic and requires a great deal of background. However, we have provided one simple assembly-language subroutine for you to use in several of the experiments. There is a good reason for including this subroutine; the equivalent function is not readily available in Applesoft. The function required is the logical ANDing of 8-bit bytes. The logical AND in Applesoft is simply a true-or-false AND operation, and it cannot be easily used for bit ANDing. The assembly-language subroutine also provides you with an introduction to how such routines can be accessed by a BASIC-language program. We have chosen to use the more complicated USR(X) command, rather than the CALL command, since we think that more will be learned.

We found that there were some limitations to the Apple. For example, there is no simple "rounding" command that can be used to round a number to a specific number of decimal digits, for example

4.1986 to 4.20. Likewise, the absence of a bit-by-bit ANDing command was a limitation that was overcome with an assembly-language routine. We also found that the potentially useful WAIT command that is used to test individual bits will "hang up" the computer if the condition is not found. The computer continues to wait if the condition is not met, and you must reset the computer to get your program going again. A color display and nice graphics are available, although we used a black/white monitor in our system.

Most of the special purpose chips, such as the analog converters, have been chosen because of their simplicity, low cost, and availability. This is not meant to be an endorsement of these products. As your interfacing sophistication increases, you will find other special-purpose devices that can serve the same function, but perhaps with added features, more resolution, different power supplies, etc. Our aim is to get you started, and not to provide you with a sourcebook of every possible interface to the Apple computer system. An impossible task in any case.

If you are interested in some additional reading about more advanced topics, we recommend:

*6502 Software Design* (21656).
*Programming & Interfacing the 6502, With Experiments* (21651).
*Microcomputer-Analog Converter Software and Hardware Interfacing* (21540).

We also recommend *TRS-80 Interfacing, Book 2.* While written around the TRS-80 computer, this book details more advanced interfacing topics such as driving high-current/high-voltage loads, serial communications, remote control, analog converters, filtering and data processing, and other interesting topics. You will quickly see that the similarities between the TRS-80 and Apple are much greater than their differences. Control signals and BASIC commands are almost identical. All of the books noted above are available from Howard W. Sams & Co., Inc., 4300 West 62nd Street, Indianapolis, IN 46268.

The pin configuration figures used in most of the figures, unless otherwise noted, are provided through the courtesy of Texas Instruments, Incorporated. The names Apple and Applesoft are trademarks of Apple Computer, Inc., Cupertino, CA. The name TRS-80 is a registered trademark of Radio Shack.

We hope that you enjoy this book, and that it leads you to design and build some interface circuits of your own.

JONATHAN A. TITUS, CHRISTOPHER A. TITUS and DAVID G. LARSEN
*"The Blacksburg Group"*

# Contents

## CHAPTER 6



## CHAPTER 7



## APPENDIX A



## APPENDIX B



## APPENDIX C



## APPENDIX D



## APPENDIX E

CHAPTER 1

# 6502 Processor

The Apple II® (Apple®) computer system by Apple Computer, Inc., uses the 6502-type of microprocessor integrated circuit. This "chip" forms the heart of the central processing unit (CPU) of the computer, the place where the actual mathematical, logical, decision-making, and other operations take place. The 6502-type microprocessor chip is manufactured by MOS Technology (Norristown, PA 19401), Rockwell International (Anaheim, CA 92803) and Synertek Corporation (Santa Clara, CA 95051).

The 6502 is an 8-bit processor. Thus, all of the mathematical, logical, data transfer, input and output operations operate on eight binary bits at a time. Each bit, of course, can be either a logic one or a logic zero. The 6502 uses an 8-bit data bus to transfer information between itself and various memory locations and input/output (I/O) devices such as a keyboard, printer, etc. In cases where the value of the information exceeds the limit of eight bits, multiples of 8-bit data words are used. Each 8-bit data word is generally referred to as a *byte*.

You should realize that the maximum value that can be expressed with eight bits is $11111111_2$ or $255_{10}$. If larger values are to be operated on in an 8-bit computer system, then multibyte operations are required. Generally, this means that corresponding data bytes in two data words are operated on, followed by the operation being performed on the next corresponding set of bytes in the large data words. In this way large values, beyond the value of 255, may be readily processed. It is important to remember, though, that the

---

Apple and Apple II are registered trademarks of Apple Computer, Inc.

Apple CPU can only process and transfer eight bits or one byte at a time.

The 6502 uses a single set of eight pins to make the connection with the data bus in the computer. This data bus is used to transfer information both to and from the computer. This type of a bus is called *bidirectional*, since it allows information to flow in two different directions. This is much like a highway that is used to allow vehicles to drive one way in the morning and to allow vehicles to travel in the opposite direction in the evening.

The 6502 generates control signals on the integrated circuit that are used both internally and externally to supervise and manage the flow of information on the bus, in one direction at a time. We will explore the generation and use of these signals later in this book.

## MEMORY

All computer systems have some memory associated with them. In general, the memory is used to store both a program that will control the operation of the computer, as well as the information that is to be processed. In the 6502 computer, each memory location can be used to store eight bits of information, or one byte of data. Most memories consist of multiples of these one-byte storage locations, generally in multiples of 1024, abbreviated 1K.

The memory locations must be addressed in some way so that the computer knows exactly where it is to store data or obtain program step information. The 6502 microprocessor chip has 16 address outputs allowing it to specify any one of $2^{16}$ or 65,536 memory locations, each of which can contain one byte. This is often shortened to 64K, indicating that 64K *bytes* of information can be addressed. In almost all microcomputer memory systems, each memory location is uniquely addressed with a 16-bit address.

The address bus lines are labeled A0 through A15, corresponding to the least-significant bit (LSB) through the most-significant bit (MSB), respectively. The LSB and MSB can both be either a logic one or a logic zero, but their *position* gives the LSB a *value* of zero or one and the MSB a *value* of zero or 32,768. Since the 6502 is an 8-bit processor, the address lines are frequently split into two groups of eight lines each, A7-A0 and A15-A8. The lines A7-A0 are referred to as the low or LO address, while lines A15-A8 are referred to as the high or HI address. In many 6502-based computers, the HI address is also called the *page address*, since the memory may be arbitrarily divided into 256 pages, with 256 bytes per page. The uses of the address bus will be explored further when software instructions are discussed and when interface circuits are developed. Unlike the data bus, the address bus is unidirectional, the address information flows

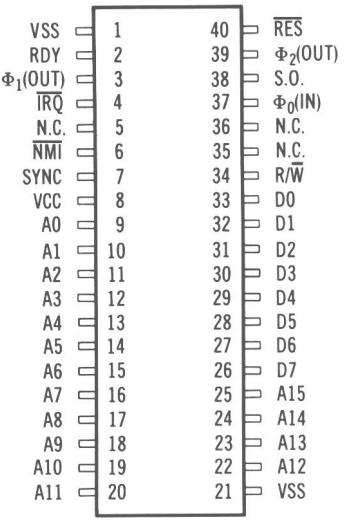



**Fig. 1-1. 6502 Microprocessor chip pin configuration.**

in only one direction, from the CPU to the memory and to external devices.

The pin configuration of the 6502 is shown in Fig. 1-1. Although most of the other signals may be meaningless to you now, you should be able to identify the 8 data bus input/output pins and the 16 address output pins.

Since the memory section is being discussed, there are two basic types of memory devices used in microcomputer systems. They are:

1. *Read/Write*—Read/Write (R/W) memory is used for the storage of data that will be changed or updated. The computer must be able to place the information in a memory location and then be able to read it back. Programs that will change are also stored in R/W memory for the same reason. The lowest cost Apple computer contains 16,384 or 16K bytes of R/W memory.
2. *Read-Only*—Read-only memory (ROM) is used when data values and program steps will not be altered. The BASIC interpreter program in your Apple system is contained in read-only memory chips. The Apple BASIC interpreter is stored in 12K of ROM.

There are various sub-classes of these types of memory devices. The R/W memories may be either *static* or *dynamic*. Static memory chips will maintain the values stored in them until they are changed. Dynamic memories require refreshing by external hardware every few milliseconds or they will "forget" or lose the data stored in them. The R/W memories in the Apple are dynamic, with the neces-

sary refreshing circuitry contained on the computer printed-circuit board.

There are many types of read-only memories. The various types are generally all static, the differences occurring in the means of storing the 8-bit values in the memory locations. The two most important types are *mask-programmed* and *field-programmed.* The mask-programmed devices have data values, program steps, etc., stored in them during the various manufacturing steps. They are generally referred to as ROMs. The field-programmable devices require some kind of special programming circuitry to store the logic ones and zeros in the various locations. Some of the field programmable ROMs, or PROMs, as they are generally called, can be erased under high-intensity ultraviolet light. They can then be reprogrammed. This is very useful when programs are being developed that will be stored in read-only memory. It does not require the development of masks and chips—an expensive process—each time a program bug is found or a change is made.

A few final words are required about semiconductor memory devices. The read-write devices are *volatile,* since data (your program and values) will "evaporate" or disappear when power is removed from the system. The read-only memories, on the other hand, are considered to be nonvolatile, since they will maintain the data or program steps (the BASIC interpreter) when the power has been removed.

Most memory integrated-circuit packages or chips do not have all 16 of the address lines connected to them. They have only enough address connections to uniquely address the memory locations within the individual chip. Thus, a 64-byte chip, small by standards of today, would only have 6 address line inputs while a 1024 (1K) byte memory chip would have 10 address line inputs. Memory chips such as these have an additional control or chip-enable input that allows banks or groups of the chips to be selected, one set at a time. Various decoding and selecting circuits may be used, thus allowing a 32K block of memory to be constructed from 64-byte or 1K byte chips, or even combinations of the two. The main point here is that the memory chips do not require all 16 address lines *to be connected directly to them,* although some combination of all 16 address bits will be used to uniquely select one byte. You should not be confused when you are confronted with a 1K $\times$ 4 bit memory that only has 10 address inputs and a chip enable input. This concept will be developed further as you study input/output data transfers.

One control signal is generated by the 6502 processor chip to control the flow of information on the data bus. This signal is noted as READ/$\overline{\text{WRITE}}$, or more simply, R/$\overline{\text{W}}$. Whenever a read, or a write, operation is to take place, the 6502 must specify a 16-bit address to

locate the memory "cell" that is to be involved in the transfer. In this case, the cell is an 8-bit word or byte.

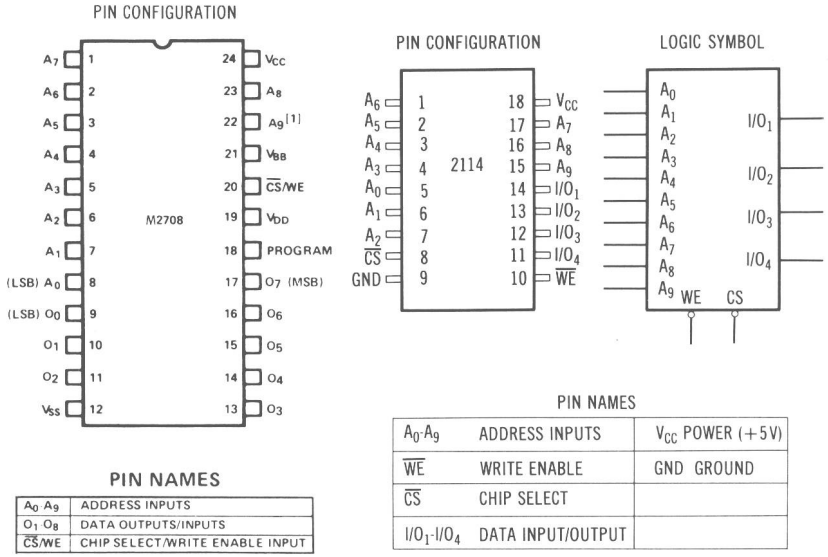
The "bar" over part of the signal notation indicates that when the signal is a logic zero, a write operation is taking place; and when in the logic one state, a read operation is taking place. Thus, a single line controls all of the memory functions. In some 6502-based computer systems and peripherals, you may see the signal "split," to provide two memory control signals, memory read ($\overline{\text{MEMR}}$ or $\overline{\text{MR}}$), and memory write ($\overline{\text{MEMW}}$ or $\overline{\text{MW}}$). This takes some additional gating, so in most cases, the R/$\overline{\text{W}}$ signal is used by itself. It is available at pin 34 on the 6502 microprocessor chip.

You may also see the notation RAM used to incorrectly signify read/write memory. The acronym RAM stands for *random-access memory*. In fact, all of the modern, easy-to-use memory devices are random access, since one may address one location and then any other, without having to sequence through all of the locations between the two addresses.

Pin configurations for typical memory chips have been provided in Fig. 1-2.

For additional information about memory devices, we refer you to

- *Intel Memory Design Handbook*, Intel Corporation, Santa Clara, CA 95051, 1975.




PIN NAMES

| | |
|---|---|
| $A_0$-$A_9$ | ADDRESS INPUTS |
| $O_1$ $O_8$ | DATA OUTPUTS/INPUTS |
| $\overline{CS}$/$\overline{WE}$ | CHIP SELECT/WRITE ENABLE INPUT |

PIN NAMES

| | | |
|---|---|---|
| $A_0$-$A_9$ | ADDRESS INPUTS | $V_{CC}$ POWER (+5V) |
| $\overline{WE}$ | WRITE ENABLE | GND GROUND |
| $\overline{CS}$ | CHIP SELECT | |
| $I/O_1$-$I/O_4$ | DATA INPUT/OUTPUT | |

**Fig. 1-2. Pin configuration for 2708 1K × 8 PROM and 2114 1K × 4 R/W memory.**

- *The 8080A/9080A MOS Microprocessor Handbook,* Advanced Micro Devices, Inc., Sunnyvale, CA 94086, 1977.
- *Mostek Memory Products Catalog,* Mostek Corporation, Carrollton, TX 75006, 1977.
- *Bipolar and CMOS Memory Data Book,* Harris Semiconductor Prod. Div., Melbourne, FL 32901, 1978.

## INPUT/OUTPUT (I/O) DEVICES

Most microcomputer-based systems are worthless without some attached I/O devices. These devices may be standard peripherals, such as card readers, printers, displays, or they may be sensors, controllers, and other devices that most people do not normally associate with computers. The Apple is no exception. It already has several I/O devices associated with it: a television display, a cassette recorder, and a keyboard.

Other I/O devices can be added to your computer. These devices may be of your own design or they may be standard, commercially available devices that are compatible with the Apple. These I/O devices are much like the individual memory locations that were discussed in the previous section. The I/O devices are attached to the data bus, since data is transferred to them and from them, and they are also connected to the address bus so that they may be uniquely addressed by the 6502 microprocessor chip.

A control signal, READ/$\overline{\text{WRITE}}$ or R/$\overline{\text{W}}$, is used to synchronize the flow of data to and from the I/O devices. This signal is also used in 6502-based computer systems to control the flow of information to and from the memory chips. Thus, there is no differentiation between memory addresses and I/O device addresses in 6502-based computers. In computers that are based upon the 8085- or Z-80-type microprocessor chips, there are different techniques that are used to address memory and I/O devices independently. Since only one synchronizing signal is used to control memory and I/O devices, the Apple's 6502 processor will be either reading or writing at all times. When the R/$\overline{\text{W}}$ signal is a logic one, the 6502 is reading information *from* the data bus. When the R/$\overline{\text{W}}$ signal is a logic zero, the 6502 is writing data *to* an external I/O device, or *to* a memory location. The "bar" over the W simply means that the write operation takes place when the R/$\overline{\text{W}}$ signal is a logic zero. You may see other signals with such bars over their names. This simply means that the signals are active in the logic zero state.

Since we will be concentrating on the use of I/O devices with the Apple, we have left a great deal of the specific discussion to the remaining sections.

### Review

At this point, you should understand that the 6502 transfers and operates on eight bits of data at a time. Complex calculations and operations often require multiple groups of eight bits or bytes. The bytes are transferred to and from the 6502 CPU on an 8-bit bus.

**Table 1-1. Control Signals Used for Interfacing**

| | | |
|---|---|---|
| **DATA BUS** | D7-D0 | An 8-bit bidirectional set of lines for transfer of information between the CPU and I/O devices. |
| **ADDRESS BUS** | A15-A0 | A 16-bit unidirectional address bus used to address both memory and I/O devices. |
| | A15-A8 | HI address bus, most-significant eight address bits. |
| | A7-A0 | LO address bus, least-significant eight address bits. |
| **CONTROL SIGNAL** | R/$\overline{W}$ | Read/write control signal. |

NOTES: The "bar" notation, i.e., $\overline{W}$, indicates a logic zero is the "active" state, the state that causes the corresponding action to take place.
In each case in which a signal is enumerated, the numbers increase as the significance of the bits increases, i.e., A15 = most-significant address bit (MSB).

The 6502 uses a 16-bit address bus to address individual memory location and I/O devices. The address bus is frequently broken into a HI and LO address bus, of eight bits each. The single control signal, R/$\overline{W}$, controls the flow of information to and from the 6502 CPU. The signals and their designations are noted in Table 1-1.

## SOFTWARE I/O CONTROL INSTRUCTIONS

### I/O Commands

The Apple computer has a number of instructions that are used to control I/O devices. For the most part, though, these instructions are used to control specific I/O devices or to perform specific functions. Without realizing it, you are already familiar with some, if not all, of these I/O instructions.

Here are some specific examples of these I/O control instructions, to refresh your memory.

The INPUT and PRINT commands are probably familiar to you. The INPUT command causes a BASIC program to stop and wait for an input from the keyboard. The PRINT command causes an answer or string of characters to be "printed" on the tv screen.

**Example 1-1. A Simple I/O Program**

```
10  INPUT "VALUE OF X IS"; X
20  PRINT " INPUT VALUE WAS"; X
```

If you executed the program in Example 1-1, the value associated with the variable, X, would have to be entered into the computer before the program passed control to statement 20. These two types of I/O statements are frequently used to allow an operator to enter a value and to see it displayed. There are many variations of both the INPUT and PRINT commands, but these two examples serve to illustrate the point; you have already been using I/O operations in BASIC-language programs without difficulty.

You may have already discovered that there are also *graphic display* I/O commands in BASIC, too. These are commands such as HOME, PLOT X,Y and SCRN (X,Y). The HOME command clears the screen, and places the blinking cursor at the "home" position in the upper left-hand corner of the tv screen. The PLOT and SCRN commands require the use of "coordinates" to indicate where an operation is to take place.

The program in Example 1-2 shows how some simple graphic display commands are used in a short program. This program generates a display of randomly changing colored dots on the tv screen. If you are using a black-and-white (b/w) tv, you will see the dots in varying shades of gray.

**Example 1-2. A Random Color Pattern Generator Using I/O Commands**

```
10  GR
20  X=INT(40*RND(1)) + 1
30  Y=INT(40*RND(1)) + 1
40  COLOR=INT(15*RND(1)) + 1
50  PLOT X,Y
60  GOTO 20
```

There are two other commands that you may not have considered to be I/O commands. These are the LOAD and SAVE commands that are used to read and store programs on cassette tapes. Each command causes a preset series of operations to take place, controlling the cassette recorder. The use of these commands is fairly obvious, so we will not provide an example.

Other I/O commands are the IN#X and PR#X operations that are associated with special I/O devices that can be substituted for the keyboard and tv display. It is important that you realize that these I/O instructions are specific to the Apple computer and its BASIC-language interpreter program. These instructions would be meaningless to other 6502-based computer systems, unless they used the Apple BASIC program. The instructions are also specific to one I/O device, i.e., the HOME command will not have an effect on the cassette recorder, *or any other I/O device*. Likewise, the INPUT command controls the input of values only from the keyboard on the console.