TAB 1449

$13.95

COMPUTER PERIPHERALS THAT YOU CAN BUILD

BY DR. GORDON W. WOLFE

COMPUTER PERIPHERALS THAT YOU CAN BUILD

BY DR. GORDON W. WOLFE

TAB TAB BOOKS Inc.
BLUE RIDGE SUMMIT, PA. 17214

FIRST EDITION

THIRD PRINTING

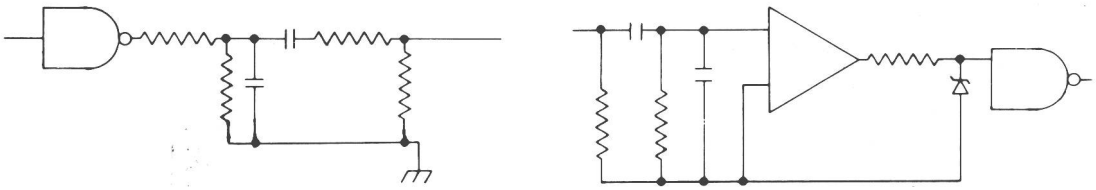
Printed in the United States of America

Library of Congress Cataloging in Publication Data

Wolfe, Gordon W.
Computer peripherals that you can build.

Bibliography: p.
Includes index.
1. Computer input-output equipment—Amateurs' manuals. I. Title.
TK9969.W64 621.3819'532 82-5688
ISBN 0-8306-2449-X AACR2
ISBN 0-8306-1449-4 (pbk.)

COMPUTER PERIPHERALS THAT YOU CAN BUILD



Introduction

With this book about computer interfacing you can add some peripheral devices to your home computer that will add to the capabilities of the computer, enhance your control over it, and increase your enjoyment of the machine. It doesn't matter what kind of computer you have. In this book you will find schemes specifically for the TRS-80, the PET, the Apple II, and all the computers that use the versatile S-100 bus and the SS-50/SS-30 bus. Yet the use of this work is not restricted to those computers. If you can find the address, data, and control-line busses in your computer and know what each of the bus lines do, you can use the devices described herein for *any* computer.

You don't have to be an electronics wizard or an electronics engineer to be able to use the book, but you will have to have some basic knowledge about electronic parts and computer programming. It is assumed that you can read a schematic diagram, and that you know what the symbols for various electronic parts are and how to build a circuit from a schematic. You should know the logic symbols for the various logic gates, such as the AND gate, the NOR gate, and the like. Appendix B contains a short lesson in assembling a circuit from parts and some of the common techniques used.

You should be familiar with your computer. at least familiar enough to know where the expansion ports are and how to program it.

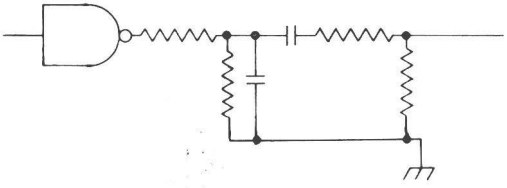
Interfacing of peripherals is best done in machine language. While it can be done in BASIC, you have to do a lot of PEEKing and POKEing, and you have to know exactly where to do it. Machine language is faster than BASIC by approximately a factor of one thousand. It will be very helpful if you have a text editor and assembler for machine-language programs and are quite familiar with machine language or assembly language for your machine. This means that you should understand the "memory map" for your computer, so that you don't program over anything the manufacturer put there on purpose. Most especially, you should know where the locations of the output or expansion ports are. Every machine is different.

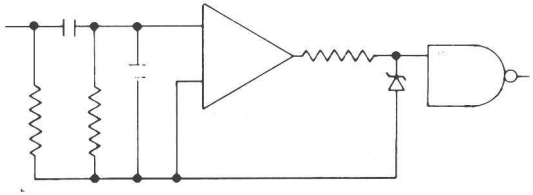
Because there are a large number of machines in use and all have different memory maps and use different processors, this book does not provide specific programs. Rather, programs are presented in a logical block-diagram form so that you can apply them to your specific computer.

I try not to use "buzzwords." Check the glossary if you come across a word you don't understand. Any language that isn't part of a general electronics background is properly defined on use. The content is presented as follows: For each of the machines described above, the book first tells you how to build a "general parallel interface" for ***that*** machine. After that, all peripherals are connected to that general interface, so there is no need to build a different set of interfaces for every machine.

Lastly, this book tries to use integrated circuits that are commonly available. Almost all the "chips" and parts used in this work are available from large distributors like Radio Shack or Jim-Pak. When a less common chip is used, the addresses of at least two suppliers for it will be given.

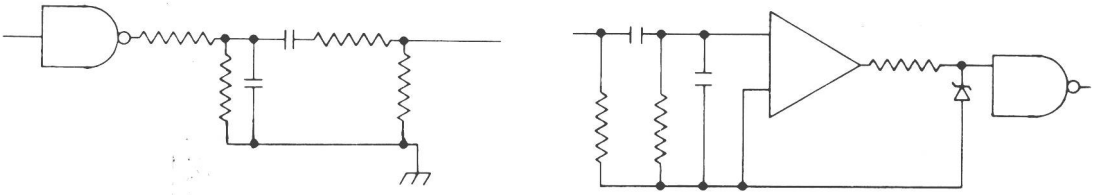
Almost all of the peripheral devices in this book can be built for under $50.00 in only a few hours time. They use ten or fewer integrated circuits. I've built almost all of them myself and have used them on my own machine, the SWTP 6800, an SS-50 bus computer.





Contents

Chapter 1



What Is a Microcomputer?

In 1973, a remarkable thing happened. It is entirely possible that future generations may look back to that date and say, "Modern history begins here." In 1973, the Intel Corporation came out with a new device called a *microprocessor*. This was a large-scale-integration (LSI) integrated circuit (an electronics package containing many thousands of individual components) that was designed to be a *programmable logic device*. Intel called their new marvel the 8008.

Until then, builders of electronic equipment had spent thousands of man-hours designing circuits for the equipment. In the 1930s this involved high-voltage power supplies and hundreds of vacuum tubes. With the invention of the transistor in 1956, designers were able to do away with both high voltage and tubes and go to lower power and smaller packages.

Soon designers found that they were spending a lot of time designing and redesigning the same circuits over and over again for the same purposes. They found that several transistors could be made from the same crystal of silicon and kept on the same package: The *integrated circuit* was born! Soon, an integrated circuit held an entire circuit such as a *gate*, building block of logic design. Gates could then be strung together to form logic devices and create a whole device to do a specific task.

As the task requirements got more and more difficult, designers once more found that they were again and again putting the same configuration of gates together in the same patterns in order to do the same types of jobs. Manufacturers countered by putting more transistors on a chip so that patterns of gates could exist on a single integrated circuit. Thus began medium-scale integration (MSI). For example, why build a clock out of individual gates, flip-flops, and counters—why not build a clock all on one chip?

Large-scale integration, with even more components on a chip, saw the advent of whole chips being designed to do specific jobs, such as counters,

clocks, digital voltmeters, memory circuits, television display generators, and the like.

Some bright person at Intel got the idea that, because it's expensive to create a different integrated circuit for each and every new task that comes along, why not make an integrated circuit that can be made to do anything if only you set it up right. It wasn't enough to have a set of output pins on the chip to configure it for various tasks. What you really needed was to have the chips read a set of instructions and perform a set of tasks based on what those instructions said to do. This is the microprocessor.

Everyone started jumping on the bandwagon. Designers worldwide began to see how they could build devices using a microprocessor as the central element. Many other manufacturers began selling their own versions of the microprocessor. At this writing, Intel has all but dropped the 8008, is concentrating on the 8080, and has just released the 8086 and 8085. Zilog developed the Z-80, which used the same set of instructions as the 8080, but was faster, easier to use, and had a wider variety of instructions. Motorola brought out the 6800 family of microprocessors and also marketed the 6502. RCA manufactures the 1802. There are several others as well.

Microprocessors started showing up everywhere. One automatic manufacturer began installing microprocessors in its cars to monitor fuel/air mixtures, among other things.

Two unforseen things happened to change the course of electronic history. Surprisingly, both led to the same final conclusion. The first was that manufacturers of electronic devices were delighted to find that product-design times went from thousands of hours to only a few dozen hours. At the same time they were horrified to discover that their technicians and engineers were still spending thousands of hours on product development—only this time they were spending all the time trying to *program* the device to do what they wanted! Many of the technicians had never done any programming before and were trying to figure out programming methods while designing the product. The answer to this was, of course, to develop the *microprocessor trainer*, a small microprocessor with some memory and readouts upon which the technician could learn the programming language of the microprocessor.

The second development was even more significant as far as you are concerned. The MITS company of Albuquerque, New Mexico, which had been a manufacturer of teaching and scientific instruments, developed a product called the ALTAIR. It was designed to be a small, home-built computer programmable in machine language only and readable only by a set of lights on the front panel. This might have gone by the wayside were it not for two facts: First, it was expandable. Inside the case there were slots for adding more components, such as memory and *peripheral interfaces*. Second, and more important, the computer kit was described in *Popular Electronics*. This magazine appeals to the home electronics hobbyist. MITS was overwhelmed by the large number of letters and phone calls asking for information about the computer and placing orders for it.

MITS had used the microprocessor as the central processing unit in a computer, thereby inventing the *microcomputer.* It became a small device which had the computing power (if not the size and total storage capabilities) of a large IBM mainframe computer. Suddenly everyone wanted one. People who had been hounding surplus dealers for scrapped PDP-8s were now able to build and own their own computer. Hobbyists discovered microprocessor trainers, which were, in effect, small computers themselves. The age was born.

BUS-TYPE COMPUTERS

The ALTAIR computer was significant, not only because it was the first computer for public use, but also because it was a *bus-type* computer. It has a number of plugboard slots on a *motherboard,* which takes signals from one plug-in circuit board to another plug-in circuit board. The lines of conductor that travel along the motherboard from one board to another are called *bus lines*. The idea of a bus is significant because it allows the computer to be expandable. As your needs change, you can change the computer by simply adding more circuit

cards to the plug-in slots. The original ALTAIR came with a motherboard and power supply, a front panel for data input by switches and data output by flashing lights, a plug-in card containing a microprocessor and some support circuitry to make it work as a computer, and a plug-in card with a very small amount of memory (128 bytes). It used an 8080 processor.

Two other bus-type computers were developed within a year. One was the IMSAI, which used the same bus structure as the ALTAIR (and the same processor) but was specifically designed to be expandable. The power supply was huge compared to the ALTAIR's, so more boards could be plugged in. Also, it had more slots available for future use. The important advantage was that it was designed to be *compatible* with the ALTAIR. IMSAI boards would fit in the ALTAIR, and vice versa. Programs for one would run on the other. This was the birth of the S-100 bus.

The other bus-type computer marketed at the same time as the IMSAI was developed by Southwest Technical Products Corporation (SWTP) of San Antonio, Texas. It broke with tradition and used the Motorola 6800 processor. It developed a whole *new* bus structure, completely different from the 100-line S-100 bus of the 8080. Since it had only 50 lines, it was deemed the SS-50. SWTP captured a large part of the microcomputer market by offering more memory (about 16 times as much as the ALTAIR), doing away with programming switches and lights on the front panel, and putting into permanent memory a *monitor* which would allow input to the computer and printout via a teletypewriter or video terminal. In addition, they began selling their programs for ridiculously low prices ($5.00 for a BASIC interpreter), and that their computer sold for less than either the ALTAIR or IMSAI, and the race was on! Several other manufacturers, caught by SWTPs low prices, began manufacturing plug-in boards for this bus or offering programs for this computer.

As an aside, it is interesting to note that at this writing, there are hundreds of S-100 and SS-50 bus manufacturers marketing circuit boards, but ALTAIR and IMSAI are no longer manufactured, while SWTP is still going strong. The reasons have to do more with business management than the quality or desirability of the products, though.

SINGLE-BOARD COMPUTERS

While the bus-type computers were very versatile, a minimum system to begin computing was quite expensive, more than $1,000 in 1977. Many people thought that they should be able to get into personal computing for a lot less than that. Many of them turned to microprocessor trainers. Still others began manufacturing a minimum computer system on a single circuit board. IMSAI manufactured one for a short time. RCA got into this market with the COSMAC ELF. It was the KIM-1 computer, and more importantly, the Apple computer that essentially captured this market.

The KIM and the Apple were self-contained. There was no fancy (expensive) case with blinking lights, just a processor, some memory (usually 1024 bytes), a monitor, and a hexadecimal keyboard and display. The Apple had its own video-display circuitry to be hooked up to a TV set for readout and a connector for a keyboard. Both could be connected to a small cassette recorder for permanent storage of programs. The total price was somewhere around $650.00.

The savings was substantial, but there was one drawback: they were not easily expandable. The bus was still there, but it took some work to be able to use it.

DESKTOP COMPUTERS

In 1978, another step forward was taken for the home user of computers. Until this time, the home computer had required some knowledge of computer systems or electronics on the part of the user. Single-board computers frequently had to have power supplies constructed and added. The boards for the plug-in computers had to be addressed and configured.

Then the Tandy Corporation did what others had been planning for some time—they marketed the desktop computer. For the first time, no knowledge of computer systems or electronics was re-

quired of the user. There was nothing to assemble or wire: Just take it out of the box, plug it in, and turn it on. Tandy was selling the TRS-80, a Z-80-processor-based home microcomputer. It was destined to outsell all other computer manufacturers *combined*. Not only was the hardware insignificant to the user in that he didn't have to do anything but type on the keyboard, but the days of programming it in machine language were also gone. The TRS-80 had its own built-in BASIC interpreter, which was available for the user from the minute the computer was turned on.

Other manufacturers soon followed suit. Commodore put out the PET, which used a 6502 processor, and Apple came out with the Apple II. Instead of a single-board computer, it was now a fully self-contained desktop unit, with all the features of the TRS-80 (and then some). Furthermore, there was provision for expansion of the computer with a peripheral bus in the back! In all cases, the user did his programming in BASIC, graphics (drawing of pictures on the screen) were available, there was instant cassette storage, and provision had been made for some peripherals—such as disk drives, printers, and telephone connections—to be added later. The *expansion port* became the key to more capability of the computer. In fact, what the expansion port really is, is the computer's bus brought out to a connector so that more "boards" may be plugged into it.

This book shows you how to tie into that expansion port in your desktop computer, or into the bus of your bus-type computer, and add additional capabilities to your computer.

COMPARISONS TO LARGER COMPUTERS

There is really very little difference between the microcomputer and its larger cousins, the minicomputer and the standard mainframe. The differences are only two: physical size and word size.

Until about ten years ago, a "computer" was a behemoth manufactured by IBM, or Honeywell, or Control Data Corporation. A computer filled up a room and took a large staff of experts to run. The popular idea of a computer was several large racks full of blinking lights and spinning tape drives. The minicomputer of ten years ago did little to change that concept beyond shrinking the size of the machine to the size of a refrigerator. Microcomputers today tend to be about the size of a television set or a sewing machine. Prices are commensurate with size.

The real difference between the three computer types is the size of the *data word*, or how many "bits" the computer can process simultaneously. Mainframe computers process 32 or 36 bits simultaneously; minicomputers, usually 16, and microprocessors, only 8.

A *bit* is a single on-off state in a computer. Eight bits together form a *byte* (note: four bits together are a *nybble*). A byte is a convenient size to process because it can represent a binary number from 0 to 255, any one of 128 letters or characters, or one of 256 data instructions. Note that a minicomputer processes two bytes at a time; a mainframe processes, four, and a micro, only one. It is said that a micro has a one-byte *word,* a mini has a two-byte word, and a mainframe has a four-byte word. (Chapter 2 discusses bits, bytes, and words further.)

Some people say that the real difference between a microcomputer and the others is that a microcomputer uses a microprocessor as the central processing unit. This is not necessarily so. The Digital Equipment Corporation PDP-11 minicomputer uses a 16-bit microprocessor called the LSI-11. There are several other 16-bit microprocessors on the market.

Physical size is no guarantee that the computer is a mini or a micro, either. *All* computers have the capability of adding more memory or more peripheral equipment. (That's what this book is all about.) A microcomputer can interface with a 9-channel magnetic tape drive just as easily as a mainframe can. Hard disk drives are already common on many microcomputers. Bascially, a general rule that the microcomputer enthusiast should not forget is *a microcomputer can do anything any larger computer can do*. It may take it a little more time, but it can be done, subject only to sufficient memory and the operator's willingness to interface more equipment to it. Most peripherals that will

work with a mainframe will also work with a micro (with the right interfacing).

STRUCTURE OF MICROCOMPUTERS

All computers (including microcomputers) have the same basic internal structure. They all have a processing unit (in this case the microprocessor), memory for both permanent and temporary storage of data or instructions, and a means of sending data to or receiving it from some other device. (It has to get the data from some place and do something with it when it's done.) Figure 1-1 is a general illustration of a computer.

The central portion of the microcomputer is the microprocessor, which acts as a central processing unit. It requires a source of power and clock pulses (which in effect tell it when to execute the next instruction). As far as the computer as a whole is concerned, it acts by itself as a *bus driver*. Connected to the bus driven by the CPU are the various memory locations and input/output registers. These supply data to the bus, store data from the bus, or take data from the bus and deliver it to an outside device, called a *peripheral*. It is the delivery of information to and from these peripherals via the data bus that we are concerned with in this book.

Every microcomputer has three busses. They generally run side-by-side, so we can talk about the "computer's bus," but it is in reality three different busses. The first of these is the *data bus*. In a microprocessor system this is usually eight lines that can carry data in either direction, either from the microprocessor to the addressed device, or from the device to the microprocessor. Some busses, such as the S-100 bus, have two single-direction data busses. This increases the complexity of the system slightly, as you will need bus drivers for each of the busses and logical systems to tell when each of the busses needs to be used. In most systems, there exists only a single bidirectional data bus of eight lines, with only one "trans-

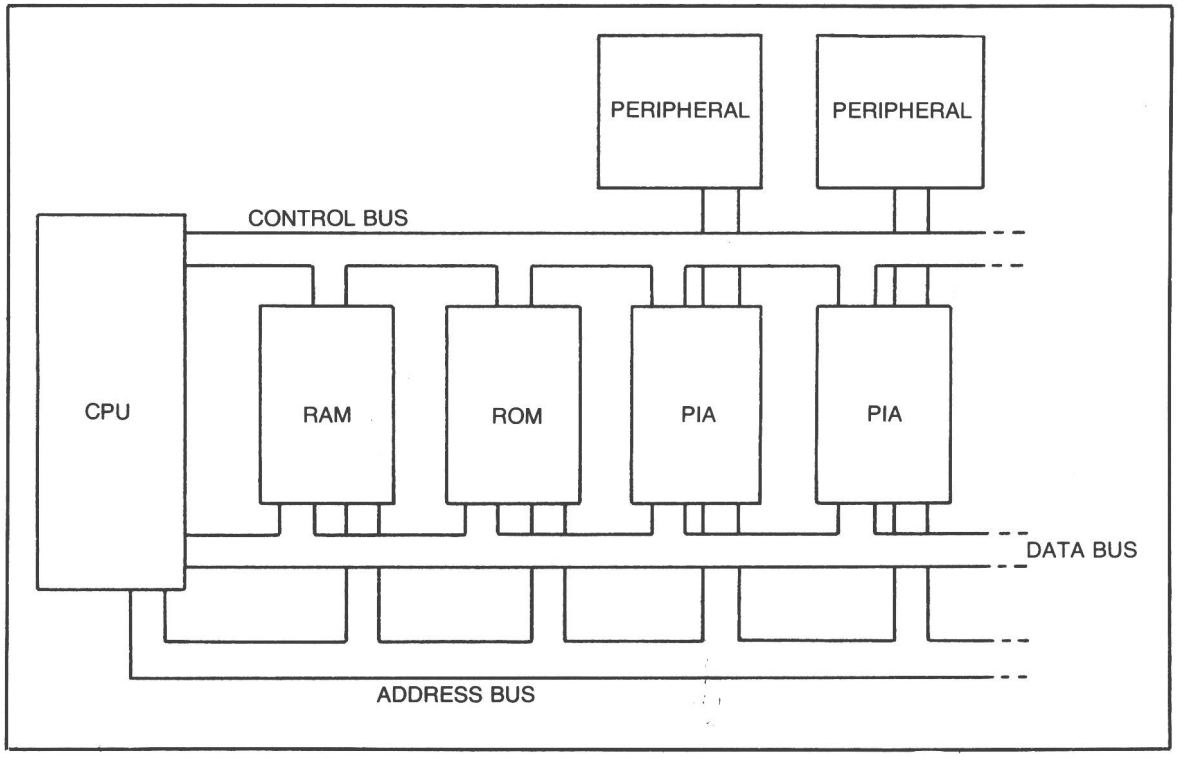


Fig. 1-1. General structure of a computer.

mitter" and one "receiver" active on the bus at a time. (More is given in Chapter 2 in the discussion on three-state buffers and bus logic.)

The second of the busses is the *address bus*. Most microprocessors use a sixteen-line address bus, although some use only an eight-line bus. Still others use eight lines as a data bus sometimes, and the same eight lines as part of the address bus. The Motorola 6809 does this. Since each of the address-bus lines can be in either a high (+5V) or a low (0V) state at any time, there are 2^{16} possible combinations of high-and-low on the address bus. This means that 65,536 possible *addresses*, or memory locations, may be accessed by the processor. Each memory location is a spot that contains, can take, or can supply eight data bits to or from the data bus. This is the *memory* of the computer. It will be very useful to you if you have a *memory map* of your computer. This tells you where in memory (at what values of the address bus) your readable memory and input/output devices are. It will be indispensable for interfacing to peripheral devices. (See Chapter 2 on in address decoding for more on memory maps.)

The third bus may be the most important of the three. It is the *control bus*. It essentially tells the other busses how and when to work. The control bus is usually 10 to 12 lines (although it may be many more, as in the S-100 bus) which tell the external components on the bus what the microprocessor wants from them. Types of signals will vary from one microprocessor to the next, but generally there will be signals such as the system clock, ground line, power lines, signals to tell the memory locations that a memory address is being looked for, signals to tell the memory whether the present operation is a read or a write operation, interrupt-request lines, requests for the processor to halt for a DMA (*direct memory access*) operation by a peripheral, confirmation by the processor that it's okay to go ahead with DMA, a signal to reset the whole system, and the like. Some busses use a line to let the memory location know that the present operation is an input/output operation, while others will use separate lines to inform the memory of read and write operations. Still others, such as the SS-50 bus, keep the serial-data-rate clock signals on the bus. Most will have at least two separate interrupt lines, maskable and non-maskable, while others will have interrupt-priority vectors. The whole thing can get quite complicated. If you are going to be interfacing to the bus, you really need to know what all those bus lines do and what their signals mean.

THE CENTRAL PROCESSING UNIT

The central processing unit has come to mean the microprocessor itself. That is, it means the LSI integrated circuit that performs the logical and arithmetic functions. However, it is actually more than that. It also includes the bus driver/buffers, the system clock, the reset functions, and the like. A microprocessor cannot function by itself; it needs some *support hardware*, just as it needs a certain minimum amount of memory to perform basic tasks.

First, though, what is a microprocessor and how does it work? Broadly, it is a programmable logic device that can obtain and execute instructions. That is, the function or logical operation that the device accomplishes may be altered by supplying additional data at its inputs. By the execution of these instructions, the microprocessor can cause the rest of the microcomputer to do the following:

1. It can input and output data in digital form. The data can be numbers, characters, or control codes. This data can be exchanged between and among the microprocessor and several peripheral devices, such as memory, printers, video displays, paper tape readers, disk memories, cassette tapes, and laboratory instruments.

2. It contains an arithmetic logic unit (ALU) which can perform arithmetic or logical operations such as add, subtract, compare, store, shift the data, perform logical AND, perform logical OR, take the complement of.

3. It can take any data from its interior and send it to a specific memory location, where it may be stored or sent to a peripheral device.

4. It is programmable; that is, it can have its function changed externally by allowing the mem-

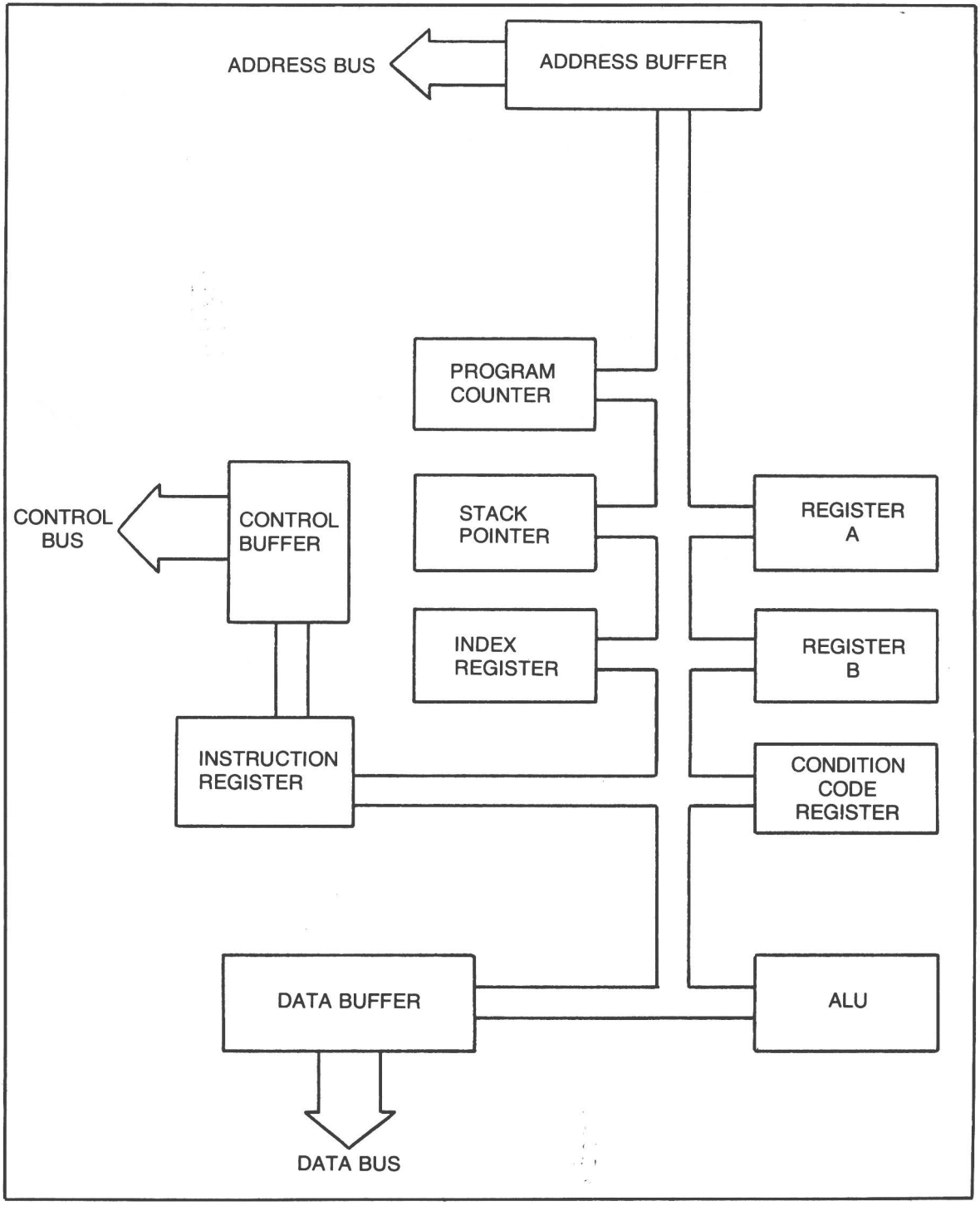


Fig. 1-2. Block diagram of 6800 microprocessor.

ory to contain different instructions. Data and instructions may be arranged in any order, in contrast to a programmable calculator, in which data goes into data registers and instructions go into instruction registers.

You might ask how does a microprocessor work? It is not strictly necessary to know how it works to be able to use it, just as it is not necessary to know how a television set works to be able to watch the news. Perhaps it might be instructive to give a general description of the interior:

Figure 1-2 gives a block diagram on the interior of the Motorola 6800 processor. Notice that the insides seem to be filled with *registers*. A register is a place for temporary storage of data. All a microprocessor does is move data from one register to another or modify the data existing in a register.

As an example of how a processor works, let's take the following three instructions of 6800 code and work through them step-by-step to see what is really happening in the microprocessor.

B6 01 07	LDA A $0107	Load register A with the contents of the memory location whose address is 0107 (hexadecimal).
8B 06	ADD A #$06	Add 6 to what is in A.
08	INX	Increment the index register by 1.
B7 00	STA A 0,X	Store register A in the memory location given by the value in the index register.

The way the processor will do it is this:

LDAA $0107	Increment the program counter. Put the value of the program counter into the Address register. Send a "read memory" request, and get the information from the data register. Increment the program counter. Put the value of the program counter into the address register. Put out a "read memory" request, and get the information from the data register. Put the two bytes of data just received into the address register. Send a "read memory" request, and get the data from the data register. Put the data from the data register into accumulator A. Increment the program counter and get the next instruction from memory.
ADDA #$06	Increment the program counter. Place the contents of the program counter into the address register. Send a "read memory" request and wait for data from the data register. Add the contents of the data register to the A register. Increment the program counter and get the next instruction.
INX	Add one to the index register. Increment the program counter and get the next instruction.
STAA 0,X	Place the index register into the address register. Place register A into the data register. Increment the program counter. Send a "write data" request. Increment the program counter and get the next instruction.

The act of getting the next instruction involves placing the program counter onto the address register, sending a "read data" request, getting the data from the data register, and placing it in the instruction register.

All this seems like an awful lot of work, but it only takes 17 machine cycles, which is about 17 microseconds. Note that all that is happening is the data is being moved around from one register to another or is being modified while in a register. It is the control-bus signals that tell the rest of the

computer what to do. The address register is connected to the address bus, and the data register is connected to the data bus.

Please note also, that a location in memory may contain either data or instructions. The machine can execute an instruction to get an instruction from a specific location in memory, and that instruction may tell it to work on some data at another memory address. The data itself may consist of a number in any of several formats. It may be a character in any of several formats which may, for example, be part of a letter that is being written on the computer keyboard (to be printed later on a typewriter), or it may be some condition code that is meaningful only in the context of the particular program being run at the time (such as, if this location contains a one, print the results on the printer, if it is zero, put the results on the video display).

I mentioned above that the address and data registers were connected directly to the address and data busses. When a read or write is commanded from the CPU, that is meant to state that "an address is on the address line—look at it, and if you are the address in question, read (or write) data from (to) the data bus." The problem with most microprocessors is that they are usually rather low-power devices. These low-power devices have to interface to many chips, sometimes dozens or scores, that read the information on them. This process is called *fanout.* Microprocessors are capable of delivering what is called "one TTL load," which is a way of saying that they can interface to one TTL integrated circuit without affecting either the voltage or current characteristics of the output. Once you try to interface the microprocessor to several chips, which are all likely to be TTL integrated circuits, you have trouble.

That is why there are special integrated circuits called *bus drivers* that take the "one TTL load" for the microprocessor and increase the capability of its bus lines. Some of these integrated circuits are capable of driving a total of 30 TTL loads, or of having 30 TTL integrated circuits "listening in" on the bus. If you need more than 30, just space the drivers down the bus every so often. For address-bus lines, the bus drivers need to work only in one direction. The microprocessor tells, via the address bus, what address is to be accessed. For data-bus lines, depending on the type of bus, you may need bidirectional drivers, that is, drivers which will send information either to or from the processor.

MEMORY

The very concept of the data processor, which executes one instruction after another, assumes that the instructions and the data upon which to work are stored in a place that the processor can get to easily. Earlier I talked about the continuous fetch and execute cycle. The CPU is continually fetching instructions and then executing them on data, both of which are brought in from external memory. This important concept of the CPU relying on memory cannot be overstressed. The CPU is a very important and powerful part of a microcomputer, but it cannot work without memory. The CPU needs to be told what to do, when to do it, and what to do it with. The arrangement of instructions in the memory is the *software*, or the computer program. The instructions or data are entered into memory, not by the CPU, but by the computer programmer. The CPU can only interpret and carry out these instructions and act upon the data in memory. The CPU does *not* contain the instructions or data, but only looks for them in memory.

How does the computer know what part of the memory to look for its instructions in? The arrangements of highs and lows on the address bus, as well as control signals that tell the computer that the CPU is looking for a specific memory location, tells the computer memory that a fetch or write cycle is being processed. Each individual memory location asks itself, "is it me?"—"is it me?"—"is it me?" Only the one that corresponds to the same patterns of highs and lows can answer, "YES!—It's me!" and will either place its contents on the data bus, or change its contents to correspond to what is on the data bus. the "Is it me"—"yes, It's me" cycle is *memory address decoding*, and the whole process is called a *memory read/write cycle.*

Note that with 16 data lines, there are 2^{16} possible memory locations, or 65, 536 possible data