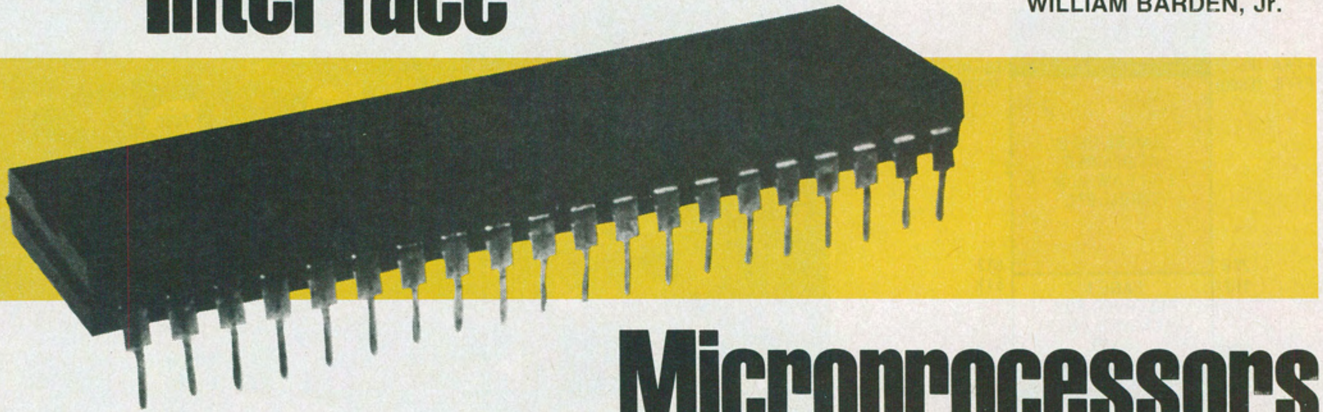
Interface Microprocessors

WILLIAM BARDEN, Jr.



You're sure to find the information provided here a great help in designing your own microprocessor-based circuits.

MICROPROCESSORS ARE BEING USED IN A variety of control applications from lumber grading to automatic bartending devices. State-of-the-art microprocessor IC's are easier than ever to interface to the external world. This article describes how to interface several types of popular 8-bit microprocessor IC's to provide TTL inputs and outputs, or to control relays or high voltage devices. The information you find in this article will be essential if you decide to design your own microprocessor-based projects.

Microprocessor structure

Figure 1 shows the general structure of an 8-bit microprocessor. The CPU (*C*entral *P*rocessing *U*nit), more commonly called a microprocessor, communicates with external memory and I/O devices along a bidirectional 8-line data bus. Instructions of one to four bytes are entered into the CPU from external memory along the data bus. The CPU decodes the instructions by executing one or more *machine cycles*, which comprise a complete *instruction cycle*. During the instruction cycle, operands (an operand is the quantity upon which a mathematical operation is performed) can be transferred between the CPU and memory or between the CPU and I/O devices. All data transfers are 8 bits, or 1 byte, at a time. The machine cycles are synchronized by a one-phase or two-phase clock generated either outside or inside the CPU IC.

External memory is addressed by a 16-line address bus output from the CPU. At certain times within the instruction cycle, that address bus holds a valid memory address. The address represents the unique memory location to be read for the next instruction or data byte, or the memory location that it is to be written into. External memory will perform a read or write when it receives a *valid memory-address* signal, the 16 bits of memory address, and a signal indicating whether a read or write is to be performed.

Input/output (I/O) devices are addressed by the CPU in two modes: memory-mapped I/O and I/O-mapped I/O. Memory-mapped I/O is used on the 6800 and 6502 microprocessors. In that mode, the I/O device is addressed exactly as a memory location is addressed, and the same signals are used to determine when the data and address output are available. In that method, the 65,536 allowable addresses on the 16 address lines must be divided between memory addresses and I/O-device addresses. Of course, the major portion of the data goes to memory addresses, because there are usually a small number of total I/O devices in the system. In the 6800 and 6502 microprocessors, consideration must also be given to page-zero memory locations, stack-memory locations, and dedicated

Photo courtesy of Intel Corporation

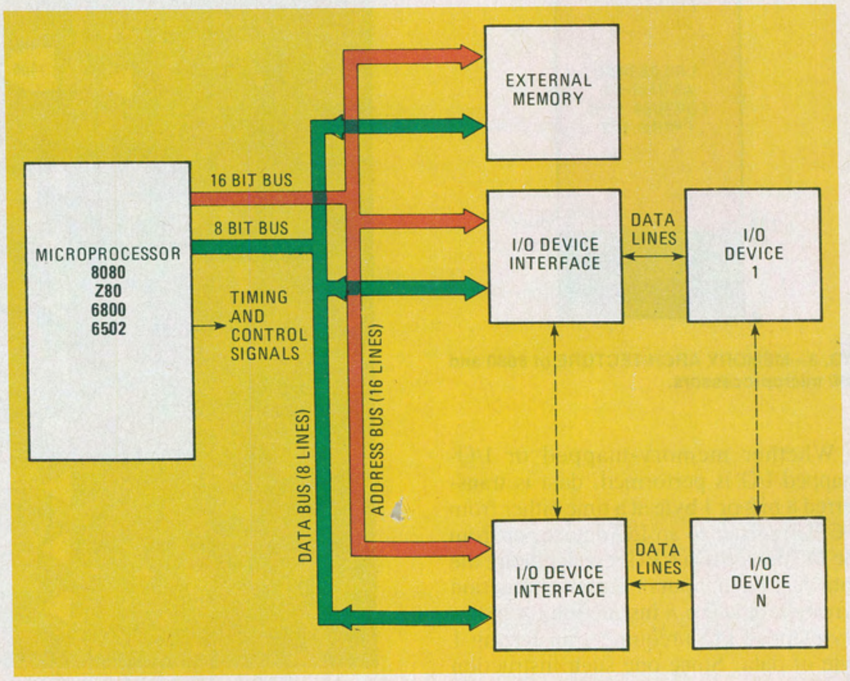


FIG. 1—MICROPROCESSOR system structure.

locations for interrupts and other system functions. Figure 2 shows general memory architecture for the 6800 and 6502.

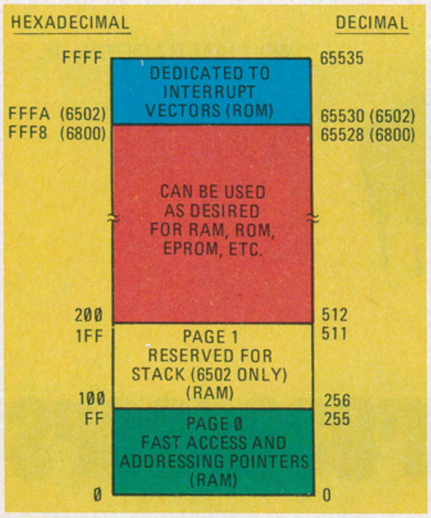

HEXADECIMAL
DECIMAL
FFFF
65535
DEDICATED TO INTERRUPT VECTORS (ROM)
FFFA (6502)
FFF8 (6800)
65530 (6502)
65528 (6800)
CAN BE USED AS DESIRED FOR RAM, ROM, EPROM, ETC.
200
1FF
512
511
PAGE 1 RESERVED FOR STACK (6502 ONLY) (RAM)
100
FF
256
255
PAGE 0 FAST ACCESS AND ADDRESSING POINTERS (RAM)
0
0

FIG. 2—MEMORY ARCHITECTURE of 6800 and 6502 microprocessors.

The I/O-mapped I/O mode is used on the 8080 and Z80 microprocessor IC's. Memory-mapped I/O can still be used on those systems, but both the 8080 and Z80 have special instructions to address I/O devices for input and output. Those instructions allow use of up to 256 different I/O addresses while retaining the 65,536 address combinations for external memory only. As in the 6800 and 6502, certain memory addresses are reserved for system functions, as shown in Fig. 3.

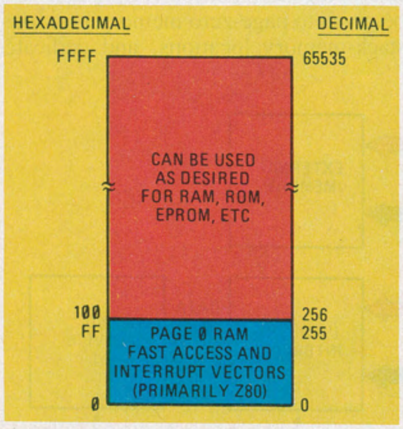

HEXADECIMAL
DECIMAL
FFFF
65535
CAN BE USED AS DESIRED FOR RAM, ROM, EPROM, ETC
100
FF
256
255
PAGE 0 RAM FAST ACCESS AND INTERRUPT VECTORS (PRIMARILY Z80)
0
0

FIG. 3—MEMORY ARCHITECTURE of 8080 and Z80 microprocessors.

Whether memory-mapped or I/O-mapped I/O is performed, data is transferred 8 bits or 1 byte at a time either from a CPU register to an I/O device, or from the I/O device to a CPU register along the data bus. Each input or output instruction (STORE A or LOAD A instruction for memory-mapped I/O) requires transferring 1 byte of data. Since one such instruction takes approximately 2 ms, and since associated instructions in an I/O loop add about another 8 ms, data-transmission speeds of up to 100,000 bytes-per-second can be accomplished be means of using register I/O of that type. Direct-money-access, which bypasses CPU registers, is a faster I/O method that allows data transfer to be limited only by

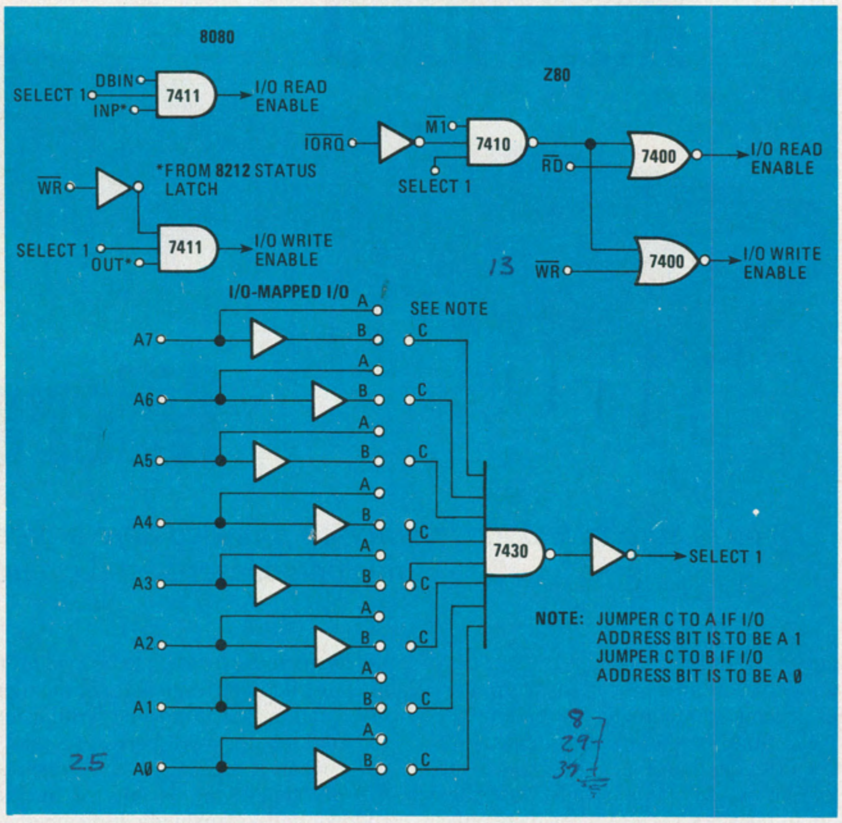


FIG. 4—I/O DECODE for the 8080 and microprocessors.

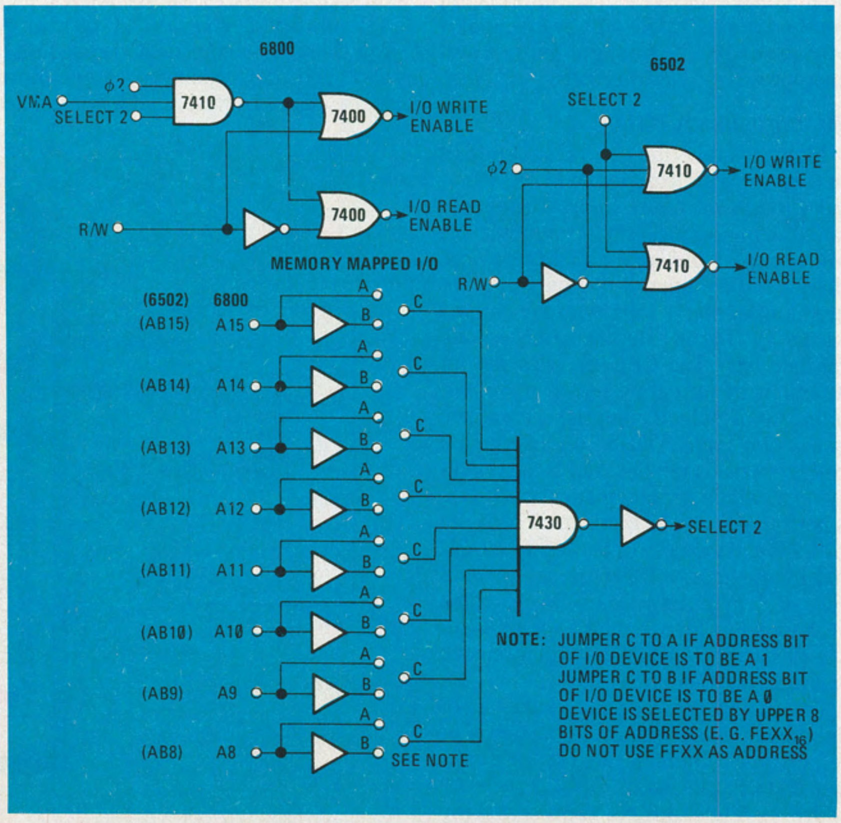


FIG. 5—I/O DECODE for the 6800 and 6502 microprocessors.

memory cycle times. In this article, however, we will consider only the simpler register I/O implementation.

In the I/O-mapped method, each I/O device has a unique device code. The microprocessor selects the I/O device by placing that code on the address bus. To transfer data between the CPU and external I/O devices requires a programmed I/O instruction, detecting the I/O-device code by decoding the address lines, and detecting control signals from the CPU that indicate when the output data is available, or when the input data should be made available. Figure 4 shows logic that is required to implement I/O reads and writes for the 8080 and Z80 microprocessors. Figure 5 shows the same logic for the 6800 and 6502 microprocessors.

Discrete I/O lines

The simplest type of interfacing consists of reading discrete-line inputs or writing discrete data into latches. Discrete data represents "on" or "off" digital data. Reading 8 data-inputs is easily accomplished by gating the input onto the data bus at the proper time. The inputs must be TTL-compatible, representing either a logic zero (a nominal 0 volt) or a logic 1 (a nominal 5 volts). If the inputs are not at TTL voltage levels, voltage-level conversion can be performed with a variety of devices, including transistors and off-the-shelf IC's.

Figure 6 shows the general method for reading eight data-inputs. The gate-enable signal is derived from the signals shown in Figs. 4 and 5 and represents the execution of a microprocessor I/O or LOAD instruction. Data on the eight lines is sampled at some time within the 2 ms or so of the I/O instruction. Gating is performed by Tri-state gates whose outputs are at a high impedance (disconnected) state when the gate-enable signal is inactive. Since the data bus is shared by memory, I/O devices and other system logic, Tri-state outputs are a necessity. Because the input data can be sampled approximately every 10 ms by software (allowing for a program overhead of loop maintenance, comparing data, etc.), the scheme can be used to sample such discrete inputs as switch closures for keyboards, burglar alarms and control functions. Switch debouncing can be accomplished by continuously sampling the input until a closure is detected, and then sampling 2 ms later or so by using a software timing loop to reject any false input caused by noise.

Data is output to the external world in similar fashion. Because the output data is present for only several hundred nanoseconds, however, it must be latched into flip-flops. Every time an I/O write instruction (or STORE instruction) is executed, new data is latched into the set of addressed I/O latches. Of course, only a portion of the data may be changed by retaining the same data in the proper bit positions of the CPU register, and changing the remainder. Figure 7 shows the method used for writing up to 8 bits of data. The clock signal that causes the data to be recorded in the latches is derived from the signals of Figs. 4 and 5.

As new data may be written out every 10 ms or so when software overhead is considered, discrete outputs of 0 to 100 kHz can be implemented. The square-wave output of the TTL latches can be used for a variety of audio applications.

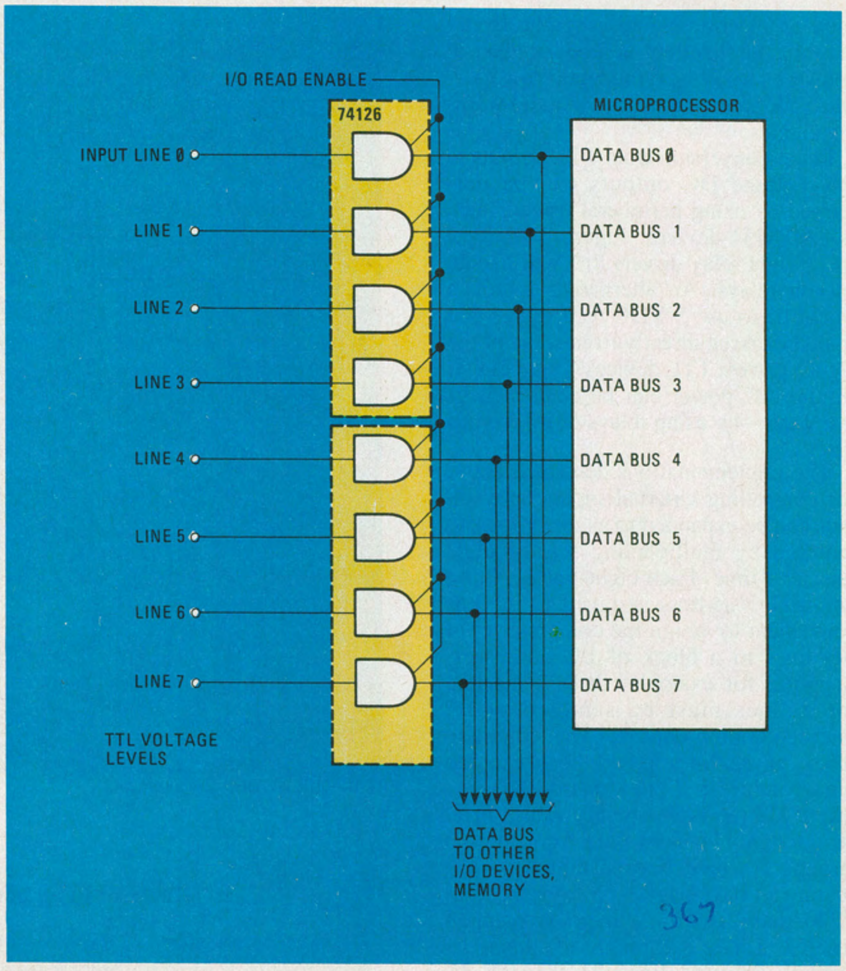


FIG. 6—DISCRETE I/O read logic.

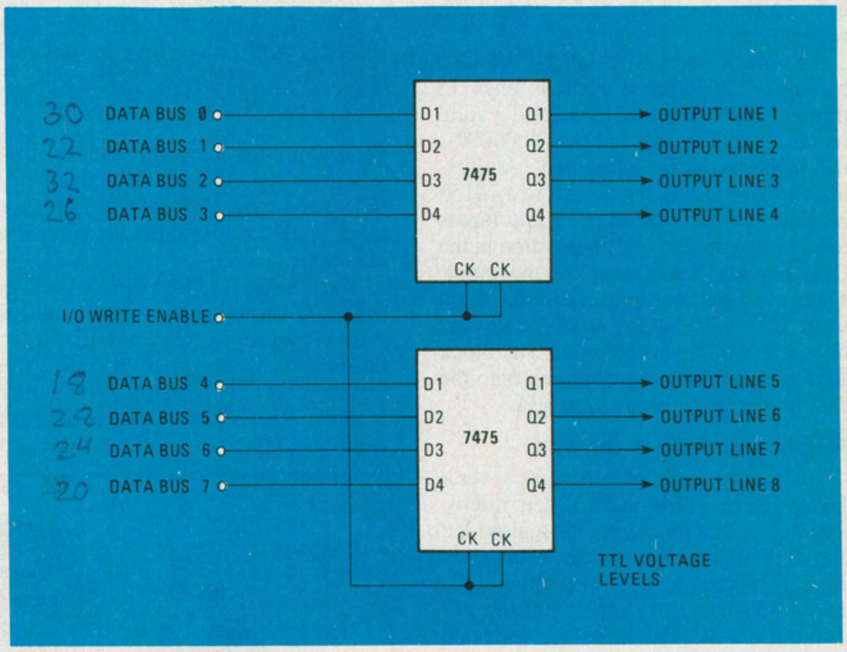


FIG. 7.—DISCRETE I/O write logic.

such as electronic music synthesis (by toggling the flip-flops to produce musical notes), as a software modulator for *Teletype* FSK applications, or for audio warning signals.

Level-conversion from TTL outputs to low-voltage DC outputs can be performed by using peripheral drivers such as LM75451 devices. Figure 8 shows a set of eight relay drivers that will handle 24-volt relays. An alternative approach would be to use 5-VDC reed relays that could be driven directly from some of the higher-current TTL devices. Devices that require AC power can be controlled in two ways—by using relays or by driving triacs.

The implementations described above for controlling external inputs and outputs can be expanded to as many lines as required by multiplexing sets of eight lines at a time. Each eight-line set has a unique I/O address assigned to it. It is convenient to assign the complete set of I/O lines to a block of I/O addresses. Suppose, for example, that 32 discrete input lines must be sampled under microcomputer control: The complete block of 32 lines might be given the binary address 11111110000000XX, where XX represents binary values from 00 through 11. Input data from lines 0 through 7 would be transmitted by executing a LOAD instruction from location FE00 (base 16) for a memory-mapped scheme. Lines 8 through 15, lines 16 through 23, and lines 24 through 31 would be addressed by LOAD instructions to hex locations FE01, FE02, and FE03, respectively.

The block address would be decoded by logic that looks at address lines 15 through 2. When those lines hold a valid memory address of 11111110000000_2, the block is being addressed. Address lines 1 and 0 are used as inputs to a 74153 multiplexer that selects one of four inputs to be transmitted to one bit position in the CPU register being used for the input. There are eight multiplexers for the eight input lines, each having inputs of address lines 1 and 0 for set selection. The block address is used to control gating onto the data bus as previously discussed.

Peripheral interface devices

Since most users may require several devices to be connected to their microprocessors, manufacturers have provided decoding, gating, latching and multiplexing capabilities on special-purpose IC's that are designed to supplement the microprocessor IC The more sophisticated IC's interface with floppy disks and video displays, while two general-purpose types are for serial or parallel data I/O. The serial devices are called USART's or UART's (or similar names), and are general-purpose (*U*niversal *S*ynchronous and/or *A*synchronous *R*eceive and *T*ransmit) devices that operate with serial data at a variety of transmission rates and in a variety of communication modes. In this article we'll consider only the type of device that provides parallel I/O since it lends itself more readily to control applications.

The parallel interface device is called a PIO or PIA (or a similar name), and Table 1 lists several. The PIA provides a set of discrete lines, ranging from 16 to 24, that can be programmed as inputs or outputs. Various other functions, such as simple hand-shaking and interrupt logic, may also be provided. The peripheral interface IC is inexpensive (typically one-half the cost of the microprocessor) and adaptable, and it provides all functions in one convenient package. Programming usually consists of resetting the device, sending out a mode-control command to prepare the device for the I/O communication mode desired, and then performing the usual I/O instructions to transfer data between the external discrete lines and the CPU register. **R-E**

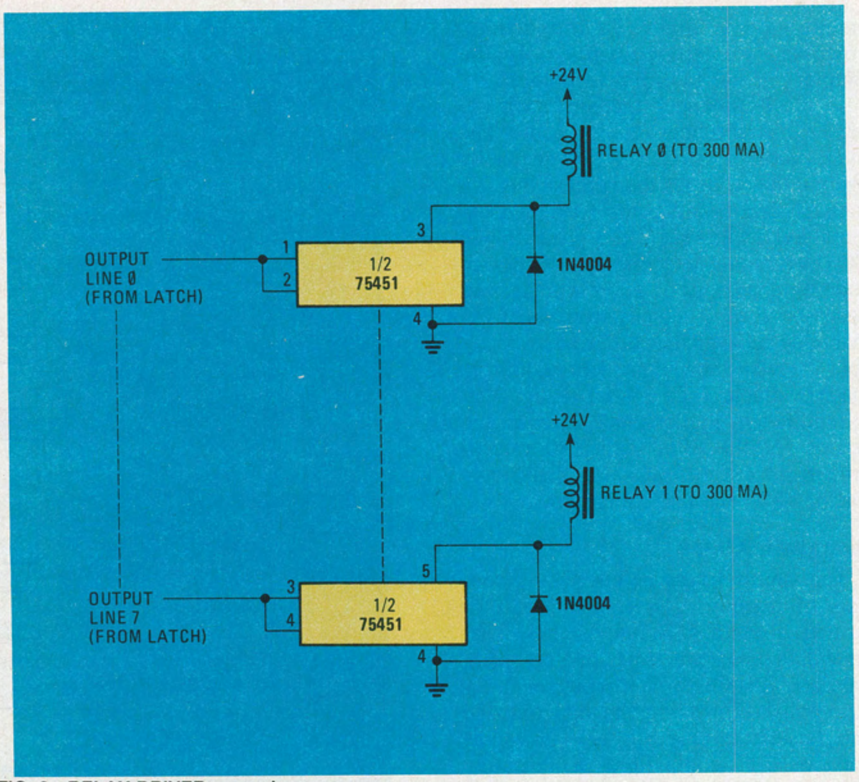


FIG. 8—RELAY DRIVER example.

Table 1—Programmable peripheral interfaces

Microprocessor	Peripheral Interface
8080 (Intel)	8255 Programmable peripheral interface
8085 (Intel)	
Z80 (Zilog)	Z80 parallel I/O circuit
6800 (Motorola)	6820 peripheral interface adapter
6502 (MOS Technology)	6520 peripheral interface device