Behavior Research Methods & Instrumentation
1978, Vol. 10 (2), 249-253

Input and output for microprocessors

STEVE GOLDBAND
State University of New York, Buffalo, New York 14226

Several alternative strategies for input and output for microprocessors are described, with examples for 8080 machines and the S-100 bus. The strategies are compared in terms of cost, complexity, flexibility, and speed in typical psychology laboratory tasks. Flag testing, interrupt processing, and direct memory access are considered in both parallel and serial modes.

The recent proliferation of microprocessors has made computing power available to more psychologists than ever before. As machines are installed in laboratories, the need for implementing various input and output devices will become more apparent. Since many microcomputer systems are designed around standard bus structures (such as the S-100), peripherals from many manufacturers may be used with a single main computer. In addition, blank prototype boards for custom circuits are available at low cost. Many psychologists, then, will need to evaluate a variety of peripherals, and some will design circuits for their own needs.

The purpose of this paper is to present an overview of some alternative strategies for implementing input to and output from microprocessors. These strategies are compared in terms of cost, speed, complexity, and flexibility for use in typical laboratory tasks. Examples are taken from a system using the 8080 microprocessor and the S-100 bus configuration. Although other machines may vary in details, much of the conceptual material cuts across various microcomputers.

MICROCOMPUTER INPUT AND OUTPUT

There are basically three strategies for accomplishing input and output (I/O) in microcomputers (Smith, 1977): flag testing, interrupt processing, and direct memory access (DMA). Flag testing and interrupt processing can each be subdivided into serial and parallel modes. In flag-testing input, the program tests a "flag" bit while waiting in a loop. When data from a peripheral becomes available, the flag bit changes state and the program exits from the loop to accept the data. In interrupt-processing input, a main program is free to execute until the data is available. At that point, an "interrupt" line on the processor is activated by an external device, and the program branches to a separate "interrupt service routine" to accomplish the transfer. When the transfer is complete, the main program resumes at precisely the point at which it was interrupted. DMA devices accomplish I/O by sharing the memory with the processor. When data is available, a processor line called "hold" is activated, and the DMA device "borrows" the system memory to make the actual transfer. Like interrupt processing, the main program is unaffected, but there is no software support in DMA. Instead, logic wired into the DMA device itself determines how the transfer is to be accomplished.

Output processing is handled similarly. Usually the computer outputs to a relatively slow device such as a printer. Consequently, the computer must wait for the device to complete each operation before sending it the next command. A flag-polled output driver waits in a loop for the peripheral to send a "done" signal, indicating readiness for the next command. An interrupt-driven output interface sends a command and is free to execute a main program while waiting for the peripheral to complete its operation. When the peripheral is ready, it interrupts the processor, which in turn sends the next command.

Serial and parallel I/O modes are distinguished in terms of the hardware that actually transfers the data. In serial I/O, the data are available one bit at a time on one wire, ordered in a precisely timed series. A group of 8 bits (called a "byte") is preceded by a start bit. The serial interface (often implemented around an LSI chip called a UART, for universal asynchronous receiver-transmitter) looks for each data bit at a specified interval after receipt of the start bit. The data word is usually terminated by one or more "stop" bits as well.

Parallel interfaces, by comparison, are composed of a series of separate wires (eight in most microprocessors) where each bit of the data is made available simultaneously. A ninth wire often serves as either a "flag" or input to an interrupt circuit depending on the type of I/O being used.

FLAG TESTING

Figure 1 illustrates a portion of an assembly language program written for the 8080 that implements a flag-testing algorithm for input (Scelbi, 1976).

The S-100 hardware for this application is shown in Figure 2 (Lancaster, 1974). The interface consists of a series of eight simple switches that represent alternative responses from a subject (S1-S8). (The switches are assumed to be "bounceless.") These are connected to a

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

```
        INTERRUPT PROCESSING

PUSH B          SAVE REGISTERS
PUSH D          IN STACK
PUSH H
PUSH PSW
LDA  PTF        GET POINTER OF STORAGE TABLE
MOV  H,A        PUT IT IN REGISTER H
LDA  PTR+1      GET LOW PART OF POINTER
MOV  L,A        PUT IT IN REGISTER A
IN   01         GET DATA FROM PORT 1
MOV  M,A        PUT IN MEMORY OF POINTER
INX  H          INCREMENT POINTER
MOV  A,H        READY TO RESTORE POINTER
STA  PTR        PUT HIGH PART IN MEMORY
MOV  A,L        NOW LOW PART
STA  PTR+1      TO MEMORY
POP  PSW        GET REGISTERS BACK
POP  H          FROM STACK
POP  D          IN REVERSE
POP  B          ORDER
EI              ENABLE INTERRUPTS
RET             RETURN TO MAIN PROGRAM
```

```
        FLAG TESTING

STRT IN   0     GET FLAG BYTE
     CPI  1     COMPARE TO 1 (READY IF =)
     JNZ  STRT  IF NOT LOOP TO STRT
     IN   1     GET DATA FROM PORT 1
     RET        RETURN TO MAIN PROGRAM
```

Figure 1. Assembly language subroutines for flag testing and interrupt for the 8080 processor.

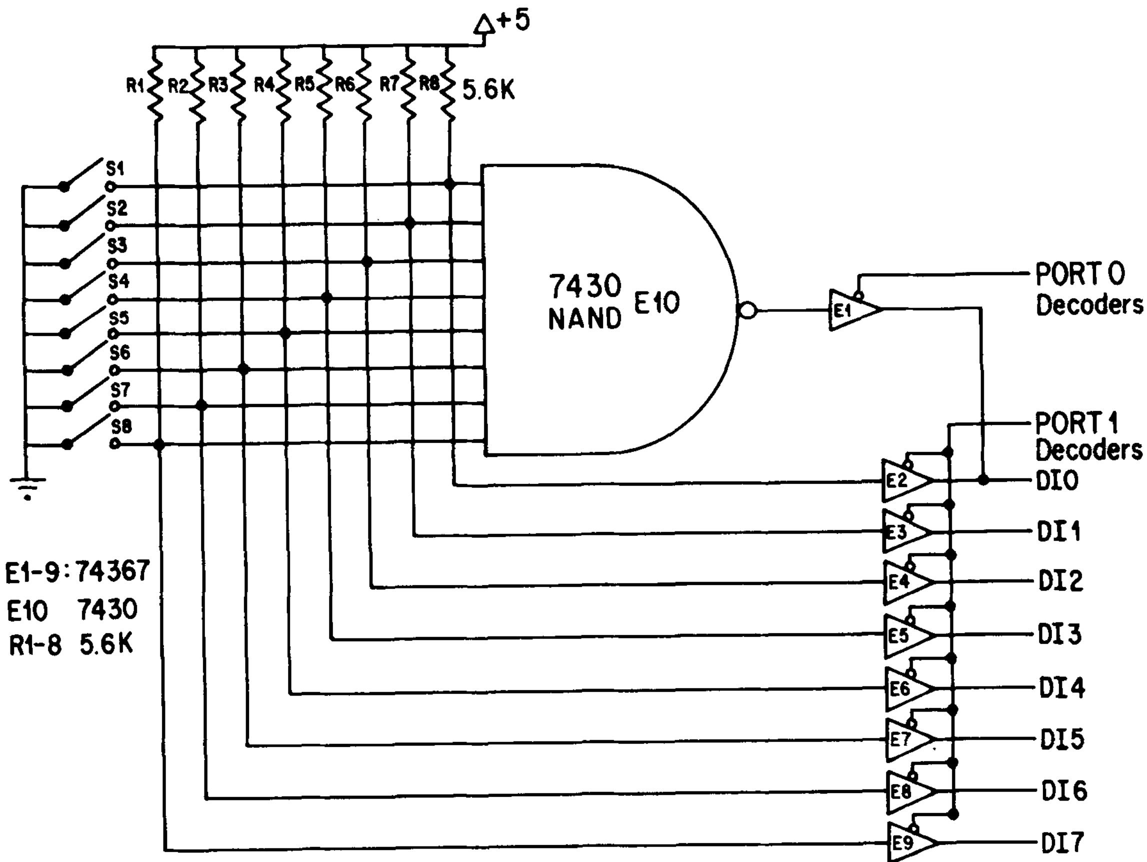


Figure 2. Parallel port hardware for flag polled input for the S-100 bus and 8080 processor.

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

"NAND" gate (E10), such that when no switch is pressed, the output of the gate is low; when any switch is pressed, the output is high. When the computer executes a "read" from input port 0, the output of the NAND gate is allowed to appear on bit 0 of the data bus by enabling a tri-state buffer (E1). When port 1 is read, the state of each of the switches is put on a separate bit of the data bus by a series of buffers (E2-E9).

When the main program is ready for input from the subject, it branches to the subroutine in Figure 1. The subroutine reads port 0 and checks for a 1 on bit 0 (data ready). If a 1 is not present, the subroutine loops. When it sees a 1, the program reads from port 1, inputting the data from the eight switches and returning to the main program with this information.

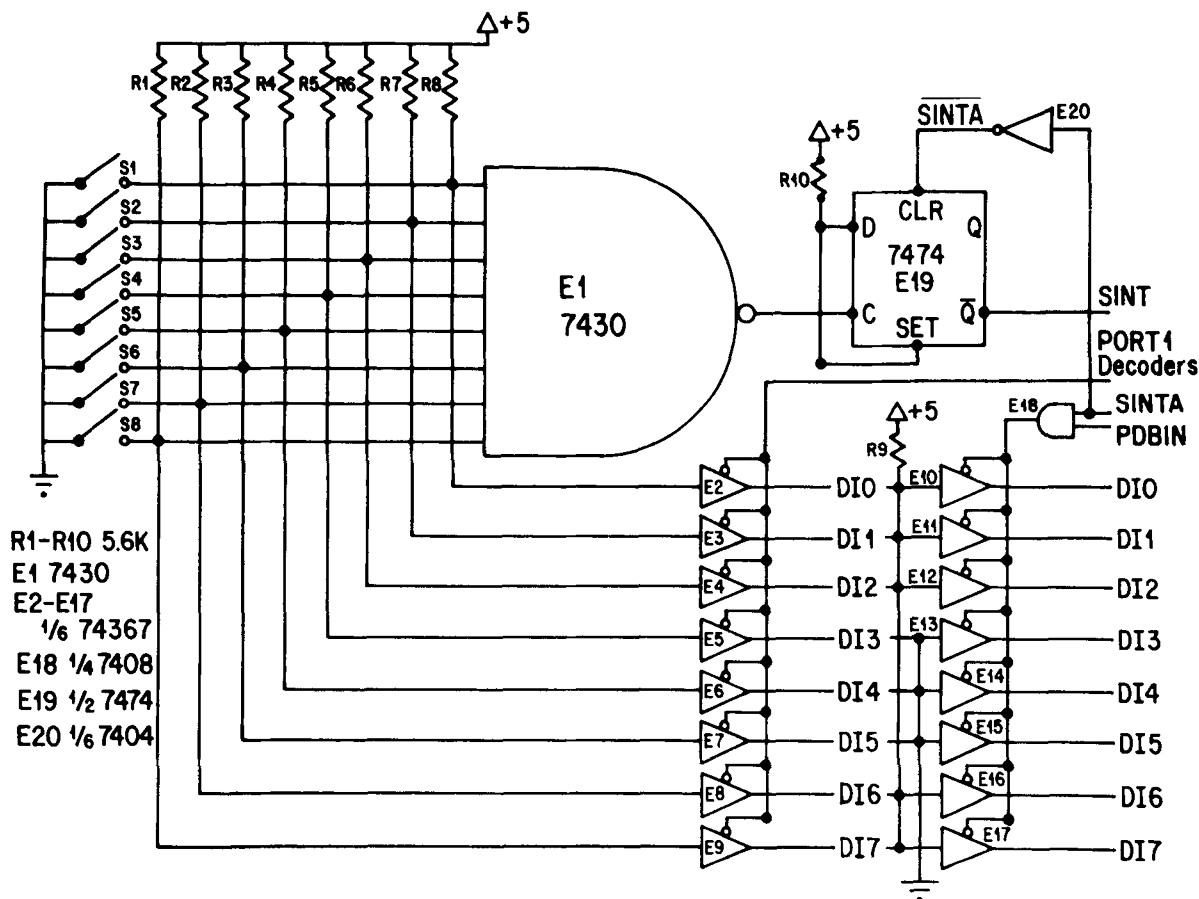
If an experiment is structured so that nothing else needs to be done while waiting for the subject's response, this method may be satisfactory. However, if the computer must continue to run a main program while waiting for the subject's response, then interrupt processing may be a better choice than flag testing.

INTERRUPT PROCESSING

An interrupt hardware interface that implements a similar function is diagrammed in Figure 3 (LaDage, 1977a, 1977b). Note that the primary difference is that the flag bit is replaced by a more complex circuit that activates the interrupt line and puts the interrupt signal on the data bus at the proper time.

When a switch is closed, the clock input of flip-flop E19 is enabled, transferring the data on the D input (hard-wired 1) to the Q and $\overline{Q}$ outputs. The resultant 0 on the $\overline{Q}$ output pulls down the interrupt line to the processor, which acknowledges with a high on SINTA (note that the software interrupt-enable flag must be set for this to take place). SINTA and PDBIN (a processor signal indicating that a data input is expected from the bus) are gated through AND gate E18 and enable buffers E10-E17, which in turn gate a RST 0 instruction onto the data bus.

The processor executes its current machine cycle, stores the program counter on the stack, and then exe-



DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

cutes the RST instruction by jumping to one of eight contiguous locations on page 0 of memory, where an interrupt service routing (ISR) must exist. SINTA is also inverted by E20 and clears flip-flop E19 in readiness for the next interrupt.

Note the increased complexity in the interrupt compared to the flag-polled software presented in Figure 1. This is necessitated by the fact that the ISR must maintain the integrity of all registers by first PUSHing them on the stack, and then POPing them off before relinquishing control. In addition, the 8080 always disables further interrupts when it is interrupted (to avoid confusion of signals), so that the programmer must remember to enable interrupts somewhere in the ISR to await further signals. Other cautions in using interrupt software arise from the fact that many programs (especially BASIC interpreters) ordinarily begin on the first memory page, so that the programmer must patch them around to use interrupts with the 8080. Furthermore, some BASIC interpreters use the "stack" in such a way as to make interrupt processing virtually impossible; check with the manufacturer of your software to be sure it is compatible before adopting an interrupt interface.

Some advanced systems are structured around many interrupt devices, each of which has a priority and can interrupt other devices of lower priority. Both the hardware and software complexity of such systems rise considerably above the simple example presented here. However, these applications of interrupts can dramatically increase the processing power of a microcomputer in a complex laboratory environment at relatively low cost.

DIRECT MEMORY ACCESS

The third class of I/O devices commonly used in microcomputers is DMA. These are complex peripherals, usually designed to accomplish a particular job very efficiently. DMA devices are often actually other microprocessors that share the same memory as the main processor. Although many high-speed peripherals available to psychologists use DMA, such as floppy disk controllers and video interfaces, it is unlikely that many psychologists will design custom DMA devices for their labs. Note that no software is used in DMA, since the peripheral itself "knows" what to do with the data according to its circuitry or, in the case of a microprocessor, its program.

COMPARISON OF STRATEGIES

As in most computing applications, there are tradeoffs among these approaches that make some more suitable for a given application than others. The tradeoffs are on dimensions of cost and complexity, flexibility in performing a variety of tasks simultaneously, speed, compatibility across systems, and ease of implementation. Figure 4 summarizes the author's subjective ratings of the various I/O strategies for each of these dimensions.

In the hardware aspect of I/O, parallel interfaces are often the least costly, least complex electronically, and have adequate speed and flexibility for most applications. They are commonly found on plug-in modules such as analog/digital converters, real-time clocks, relay

		Hardware cost complexity	Software complexity	Software flexibility	Hardware flexibility	Speed	Capability of complex programs	Ease of Use	Examples
Flag polled	Serial	++	+++	++	++	--	--	+++	terminals, printers cassette I/O
	Parallel	+++	+++	++	+	-	-	++	fast printers, I/O boards, A/D conv.,
Interrupt	Serial	+	-	+	-	+	+++	+	terminals, time-share stations
	Parallel	++	-	+	-	++	+++	+	clocks, subject stations
Direct Memory Access		-	*	*	-	+++	+++	+	video interfaces, disk controllers

+++ Excellent
++ Good
+ Acceptable
- Marginal
-- Poor
* Not applicable

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

interfaces, speech synthesizers, some printers, and other medium-speed devices. Parallel I/O has the drawback of requiring at least 8, and as many as 20, separate wires for the connection (if both input and output are needed). Furthermore, because the signals are logic level (TTL), it is not good practice to extend them more than a few feet from the CPU. Thus, parallel devices are best suited for peripherals that are installed in or near the computer's cabinet and for those that require moderate speed.

Serial I/O methods are typically slower than parallel, and require somewhat more complex and expensive hardware. Their primary advantage is that they require only one signal wire in each direction and one ground wire. As a result, it is feasible to convert the TTL level to a more robust signal or frequency modulated tone (using a MODEM) and transmit it over a distance without degrading. Most timesharing systems on large computer systems use serial interfaced terminals. Another advantage of serial I/O is that a standard exists (RS-232) for connectors and signal levels which facilitates interconnection of microcomputers to peripherals made by a large number of manufacturers, including those of mini and full-size computers. Serial I/O is frequently used in relatively low-speed devices such as terminals, Teletypes, printers, and cassette storage systems.

DMA interfaces are electronically complex. However, they are extremely fast and require no management from the CPU. They must be tailored to the specific task and system on which they are used, and so suffer from relatively poor compatibility across systems. Because of cost and complexity, they are best suited to critical and demanding tasks that would otherwise burden the processor with their service, such as video displays and floppy disk controllers.

Flag-polled I/O software is probably the simplest and cheapest to implement. Its primary disadvantage is that it monopolizes the CPU while waiting for a peripheral, drastically reducing potential throughput. In many situations this may be quite acceptable, though, since the machine may be dependent on a slow peripheral device such as a person. Operating systems that require user commands often use flag polling, as do programs that simulate terminals, and text editors.

In many real-time applications the computer may have to continue processing while waiting for random events from the lab environment. In such cases, interrupt I/O may be the best alternative despite the complexity of hardware and software implementations. For instance, if a computer must monitor a subject's responses on several dimensions while at the same time presenting stimuli based on those responses, flag polling would tie up the machine most of the time. An interrupt system would allow computation for stimulus presentation while always being available for input. In complex interrupt systems, one machine might even service several such experiments.

Programming for interrupts can be difficult, since it must be carried out at the machine language level, and requires careful attention to all possible sequences and priorities of events and to links between high-level programming languages and machine language.

REFERENCES

La Dage, D. Interrupts exposed: Using microprocessor interrupts effectively. *Kilobaud*, 1977, **4**, 18-21. (a)

La Dage, D. Interrupts exposed: Implementing an interrupt-driven system. *Kilobaud*, 1977, **5**, 78-80. (b)

Lancaster, D. *TTL Cookbook*. Indianapolis: Howard W. Sams, 1974.

Scelbi Computer Consulting Inc. *8080 Software guide and gourmet cookbook*. Milford, Conn: Scelbi, 1976.

Smith, M. L. Build your own interface. *Kilobaud*, 1977, **6**, 22-28.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.