How to expand a microcomputer's memory

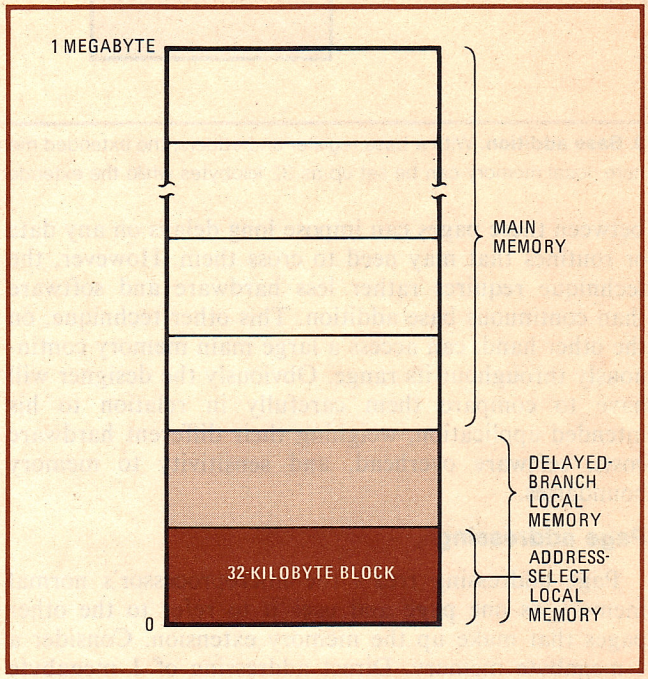
Microcomputers can address more memory than they were meant to if that memory is split into pages or if extra hardware adds address bits

by Howard Raphael, *Intel Corp., Santa Clara, Calif.*

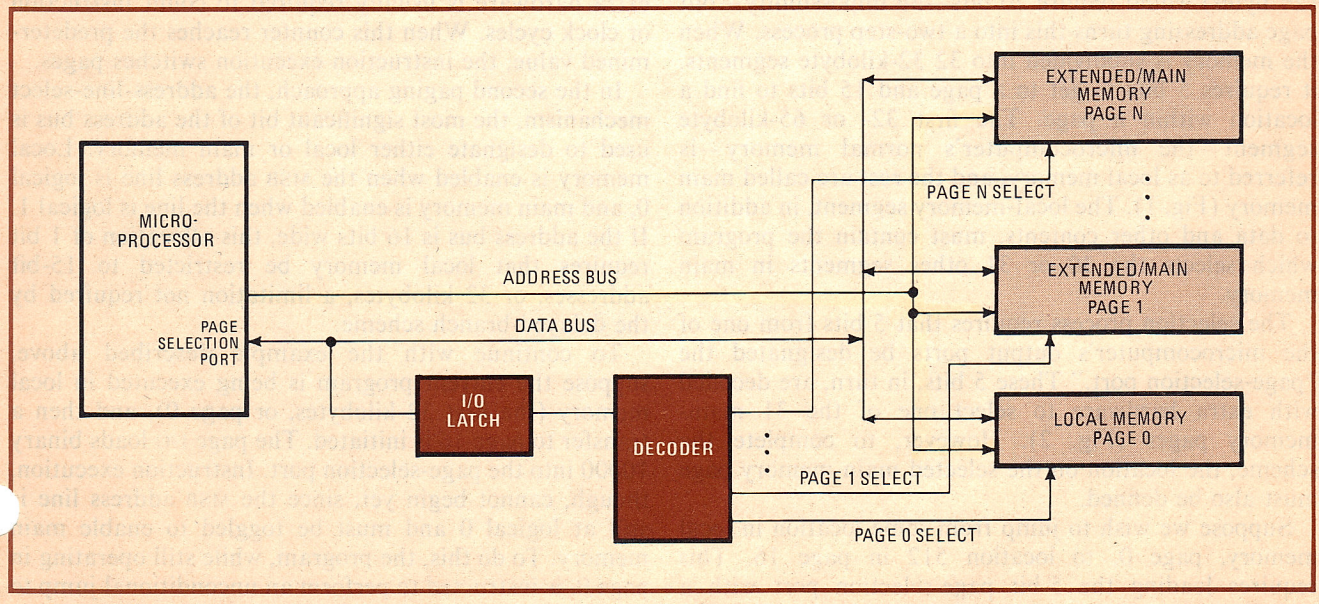
☐ Microcomputer designers are rushing to add more memory to their systems, to make room for improved software versions of existing system features or altogether new features in software form. The steady drop in memory-chip cost has triggered the rush, but a bottleneck is created by the fact that most byte-oriented microprocessors have only 16 address lines.

To address more memory than the 65,536 bytes these 16 lines can handle, two techniques are available. They are segmented paging and continuous base addition. Using these techniques, the designer can easily expand read-only memory, to, say, store the multidimensional look-up tables that can replace complex calculations and maybe even speed them up; or he can add extra ROM or random-access memory for, say, the diagnostic programs and facilities that always have user appeal.

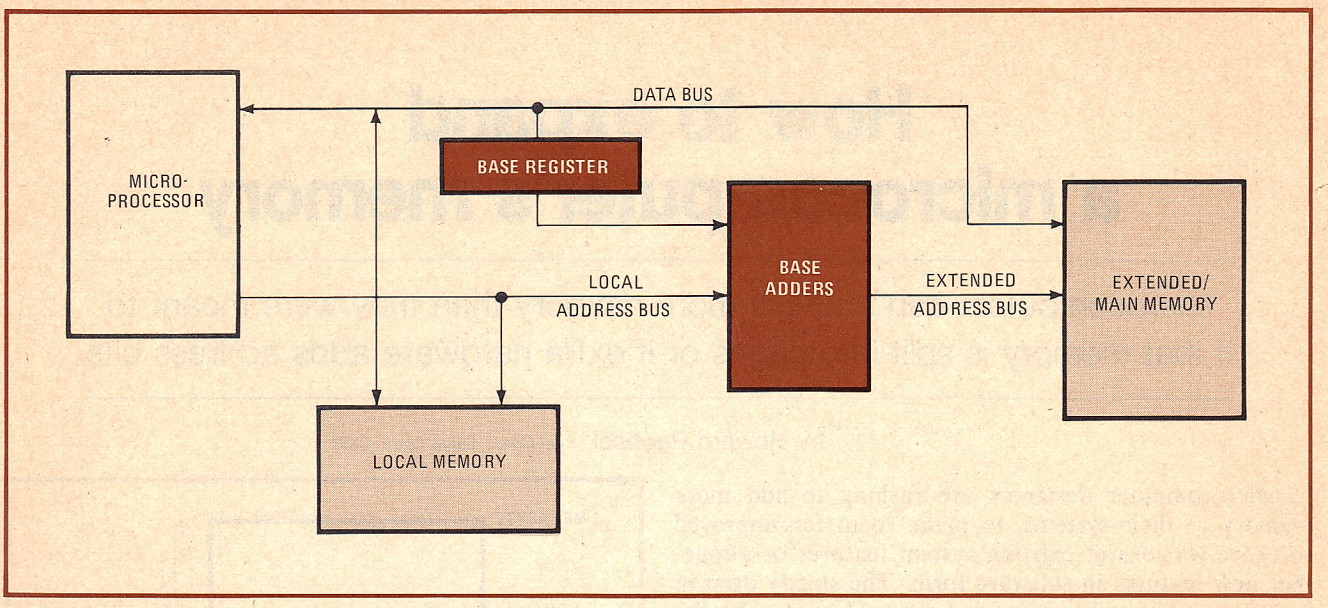
Segmented paging, as its name implies, addresses separate segments of memory, and the boundaries



1. Page addressing. The memory is segmented into 32-kilobyte pages. With the delayed-branch accessing method, the local memory can be as large as 65 kilobytes, while with the address select method, 1 address bit is preempted, reducing it to 32 kilobytes.



2. Paging memory. In the page-addressing scheme, 5 bits are taken from the microcomputer's output port, or data bus, to designate one of 32 pages in memory. The decoder then selects the appropriate page of the memory to be accessed by the microcomputer's address bus.



3. Base addition. With a base register and adders, the extended memory can be organized as a continuous 1-megabyte range, or, as shown here, local memory can be set up as 32 kilobytes while the extended main memory overlaps by 28 kilobytes.

between these pages can impose long delays on any data or routines that may need to cross them. However, the technique requires rather less hardware and software than continuous base addition. This other technique, on the other hand, can access a large main memory continuously throughout its range. Obviously the designer will have to compare them carefully in relation to his intended application, weighing their different hardware costs, software overhead, and sensitivity to memory boundaries.

Page addressing

Page addressing treats the microprocessor's normal memory as one page and uses it to refer to the other pages that make up the memory extension. Consider a 1-megabyte memory. Direct addressing of 1 megabyte (2^{20} bytes) would require 20 bits—an impossibility—but page addressing turns this into a two-step process. When the memory is subdivided into 32 32-kilobyte segments, it requires 5 bits to get to a page and 15 bits to find a location within a page. The first 32- or 65-kilobyte segment—the microcomputer's normal memory—is referred to as local memory, and the rest are called main memory (Fig. 1). The local-memory segment, in addition to data and other contents, must contain the program which selects the 30 or 31 other segments in main memory.

The selection process requires that 5 bits from one of the microcomputer's output ports be designated the "page-selection port." These 5 bits, in turn, are decoded with extra hardware to select one of the 31 main-memory pages (Fig. 2). However, to complete the scheme, the location on the selected main-memory page must also be defined.

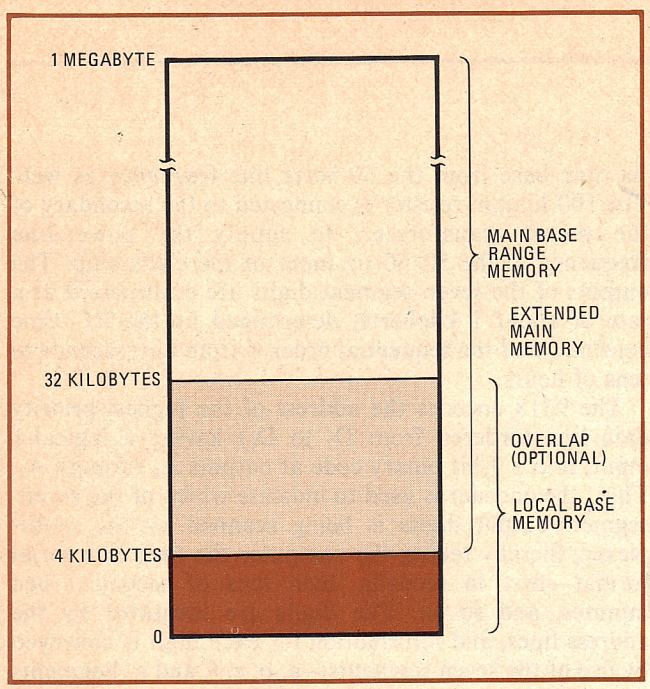
Suppose we wish to jump from some location in local memory, page 0, to location 512 in page 16. This requires loading the 5-bit page-selection port with a binary 10000 to locate page 16. On completion of that loading instruction, the program execution resumes in page 16 at the address designated by the contents of the program counter. However, we specifically wanted to be at location 512 in page 16. Thus, we would either have to arrange to be at location 511 in page 0, and insert the page-select I/O instruction there, or we need to make an unconditional jump to location 512 immediately upon entering page 16.

Neither alternative is convenient. Two solutions are possible: the delayed branch or the address-line-select mechanism.

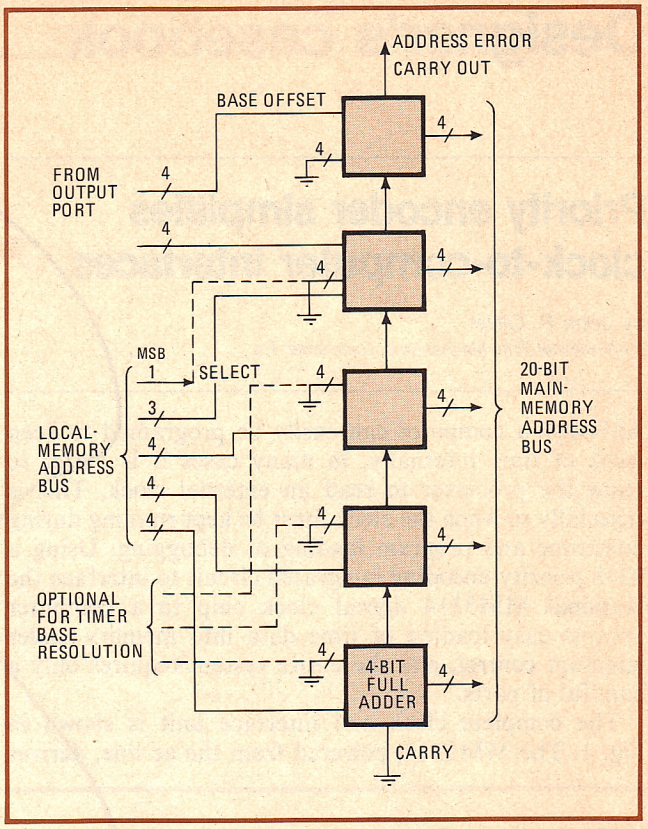
The delayed branch delays the transfer of program execution from page 0 to page 16 by enough time (one or two instruction cycles) to allow the program to execute a branch in page 0 to location 512. The page transfer then is enabled to take effect on the first instruction cycle after the branch. To implement the delayed branch, extra hardware is usually required to count instruction or clock cycles. When this counter reaches the predetermined value, the instruction execution switches pages.

In the second paging approach, the address-line-select mechanism, the most significant bit of the address bus is used to designate either local or main memory. Local memory is enabled when the MSB address line is logical 0, and main memory is enabled when the line is logical 1. If the address bus is 16 bits wide, this allocation of 1 bit requires that local memory be restricted to 15-bit addresses, or 32 kilobytes, a limitation not required by the delayed-branch scheme.

To continue with the example described above, suppose the current program is being executed in local memory (the first 32 kilobytes, or page 0), and then a transfer to page 16 is initiated. The page I/O loads binary 10000 into the page-selection port. Instruction execution, though, cannot begin yet, since the MSB address line is still at logical 0 and must be toggled to enable main memory. To do this, the program, while still operating in page 0, is instructed to perform an unconditional jump to address 32 kilobytes plus 512. Since the program counter now moves beyond the 32-kilobyte limit of page 0, its



4. Offset. For the base-adder scheme, an offset address is loaded into a base register, the contents of which are then added to the local address bus to generate a 20-bit extended address. Local memory, however, is addressed by the microcomputer's 16-bit bus.



5. Quad adders. In base addition, the base offset value is added to the contents of the address bus by five quad adders. In this example, the most significant bit of the address bus is used to signify that main, not local, memory is to be accessed.

MSB line and hence that of the bus, too, change to logical 1, placing the program counter at location 512. Simultaneously, the I/O port switches the counter to page 16.

The address-line-select scheme thus does not require the additional hardware of the delayed-branch approach, since it uses the MSB address line to enable main memory whenever necessary. However, as noted, use of the memory line does halve the usable local memory.

Both the delayed-branch and address-line-select paging schemes require the extended memory to be segmented into binary-related, unoverlapped memory ranges (pages of 4, 8, 16, 32, or 65 kilobytes). All addressing within these boundaries is relative to a particular page. For some applications, this may be acceptable and even convenient, since unoverlapped boundaries offer some measure of memory protection. But it is time-wasting and therefore unacceptable in other applications, since all excursions to page extensions must originate in local memory and cannot easily occur directly between the page extensions. Instead, switches must be between page 0 and page n and then back to page 0. If overlapping and direct switching are desirable, it is better to use the base-adder technique.

Base addressing

In applying the continuous-base-addressing technique (Fig. 3) to a 1-megabyte memory, the program loads an offset value into a separate 20-bit register through one of the microcomputer's output ports (Fig. 4). It then adds this value to the current value of the 16-bit program counter to develop a new 20-bit address for any location anywhere within the entire memory.

Since the 20-bit extended address bus will cover the full megabyte, local and main memory can be overlapped if desired. But usually it is more convenient to divide the memory into two parts: local memory of either 32 or 65 kilobytes, as before, and continuous main memory, comprising the remainder up to 1 megabyte (Fig. 3). Then the system uses local memory for normal operation and calls upon main memory only when it is needed for special data or routines or for distributed-processing communications, the advantage being that only on these occasions is the system slowed down by the extra addition stage.

To call upon main memory, the computer can do one of two things. As in paging, it can use the MSB line of the address bus to cause the switch (Fig. 5), in which case local memory is again restricted to 32 kilobytes. Alternatively, it can designate a separate flag to indicate that the contents of the 20-bit register and the program counter are to be added and used to access main memory.

Although this technique requires extra hardware in the form of the 20-bit offset register and four quad adders, it does offer a way of extending memory that is unaffected by page boundaries.

Interestingly, if only academically so, note that a full 20-bit offset address need not be used if the designer can work with page boundaries. For example, 8 bits could be used to provide the offset in the base register. The total bus width would then be 20 bits, and the base register and the 16-bit local memory bus would overlap to create steps of 4 kilobytes each. □

Memory-mapping techniques expand 8-bit-μP applications

If your design needs 8-bit computational ability and a large address space, you can combine Z80 block-I/O commands with a memory-mapper device to expand your address space to as much as 16M bytes.

John Tomaszewski, Consultant

Increasing μC-system memory space with new memory-mapping peripheral chips can greatly extend the application range of 8-bit processors. This article describes a circuit that uses Texas Instruments's 74LS610 memory mapper and an SN74LS373 3-state 8-bit D-type latch to increase a Z80's memory-address range from 64k to 1M bytes. (The LS610 can expand the Z80's memory range to 16M bytes max, but this implementation reserves four address bits for user-defined control purposes.)

Block I/O eases data transfer

Word lengths pose the biggest problem in using the LS610; it has 12-bit registers while the Z80 contains an 8-bit data bus and a 16-bit address bus. Transferring information between the devices thus requires two steps. For a Write command, for example, this design first transfers eight bits of data from the Z80 to the LS373 latch; the Z80 then decrements its data-byte counter, directly writing four bits into the middle four bits of the selected 12-bit index register in the LS610 while the LS373 simultaneously writes the other eight bits into the same register.

Achieving this seemingly cumbersome data transfer becomes easy when you use some special block-I/O instructions provided in the Mostek Z80 assembly-language instruction set. These powerful instructions —INIR, OTIR, INDR and OTDR—allow you to transmit the contents of the Z80's C register on the eight least significant bits (LSBs) of the processor's address bus while simultaneously transmitting the contents of its B register on the eight most significant bits (MSBs).

During these special I/O instructions, the Z80 also uses its B register as a data-byte counter, allowing you to transfer one to 256 bytes to or from memory. As each byte transfers, the B register decrements and outputs to the address bus; when the register goes to zero, the μP executes the next program instruction. The B register also controls the 2-step, 12-bit data-transfer sequence already described: When its value is even (B_0=0), the Z80 transmits the eight LSBs; when its value is odd (B_0=1), the μP transmits the four MSBs. Additionally, bits $B_{1\text{-}4}$ select one of the LS610's memory-mapping registers, each corresponding to 512 4k-byte pages, for data I/O.

The following example of the OTIR block-output instruction assumes that you have preset several Z80 registers. First, initialize the B register (the data-byte counter) at 00100010. Because the register's value is set to 34_{10}, the instruction transmits a 33-byte memory array (the register is decremented once upon initialization). Next, set the 16-bit HL register pair to point to the first byte in the memory array. Finally, the C register points to the I/O device (here, the LS610); set this register's value to FF_H, the highest I/O address.

Assuming these initial conditions, when the Z80 encounters an OTIR instruction, it proceeds as follows (**Figs 1** and **2**). During the first machine cycle, it temporarily stores the value in the memory location that the HL register pair points to. Next, it decrements the B register (here, to 00100001) and places this value on the top half of its 16-bit address bus with the I/O-device address (FF_H from the C register) in the

Text continues on pg 232

TABLE 1—33 BYTE-ARRAY MEMORY IMAGE

MEMORY ADDRESS	8-BIT BYTE	74LS610 REGISTER BEING ACCESSED	Z80 B REGISTER DURING I/O TRANSFER
HL	A_{19}-A_{12}	00	00100001
HL + 1	XXXX C_3-C_0		00100000
HL + 2	A_{19}-A_{12}	15	00011111
HL + 3	XXXX C_3-C_0		00011110
HL + 4	A_{19}-A_{12}	14	00011101
HL + 5	XXXX C_3-C_0		00011100
⋮	⋮	13 TO 3	⋮
HL + 29	A_{19}-A_{12}	2	00000101
HL + 30	XXXX C_3-C_0		00000100
HL + 31	A_{19}-A_{12}	1	00000011
HL + 32	XXXX C_3-C_0		00000010
HL + 33	DON'T CARE		00000001

(B REGISTER DECREMENTS TO ZERO AND THE BLOCK-I/O INSTRUCTION TERMINATES)

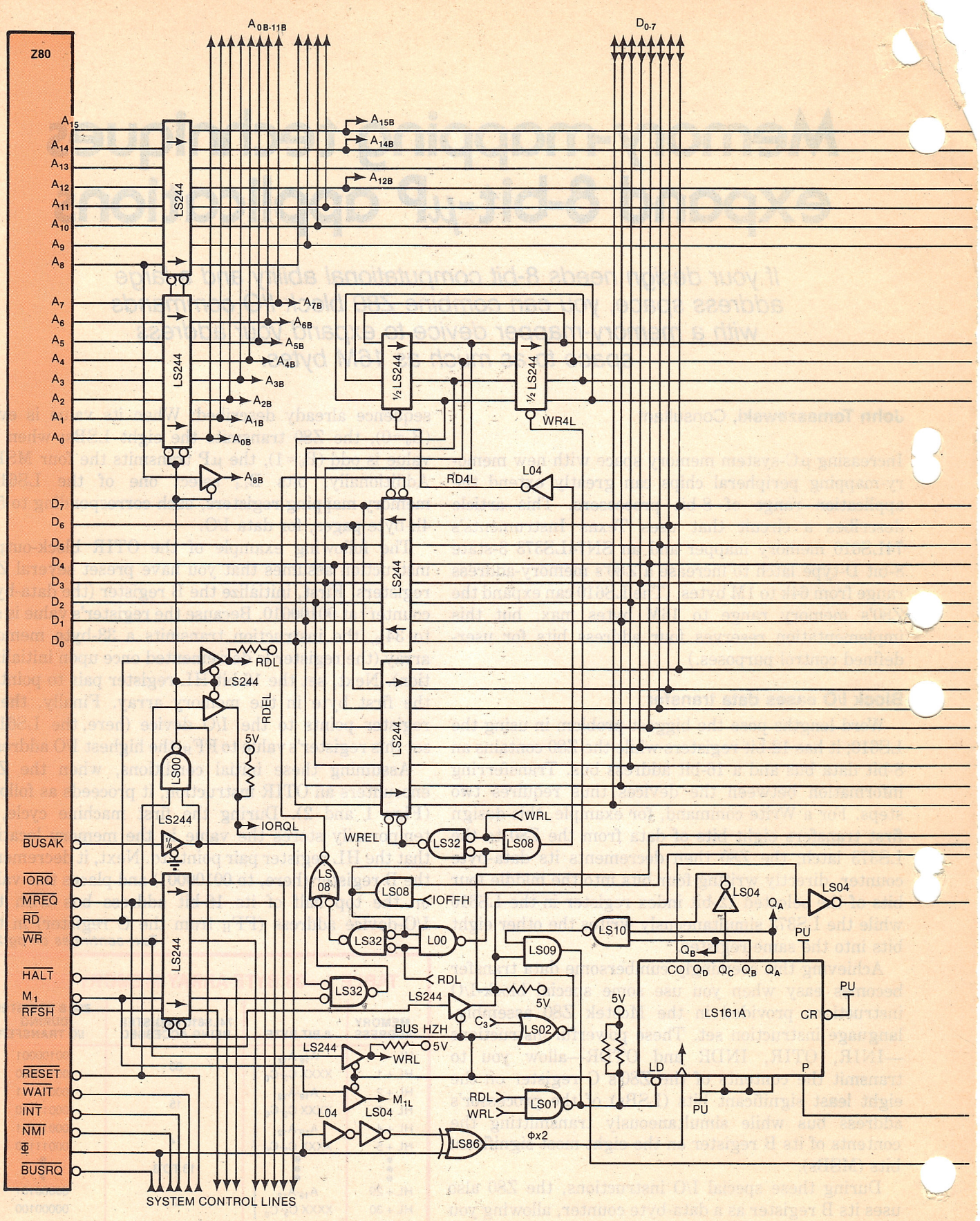
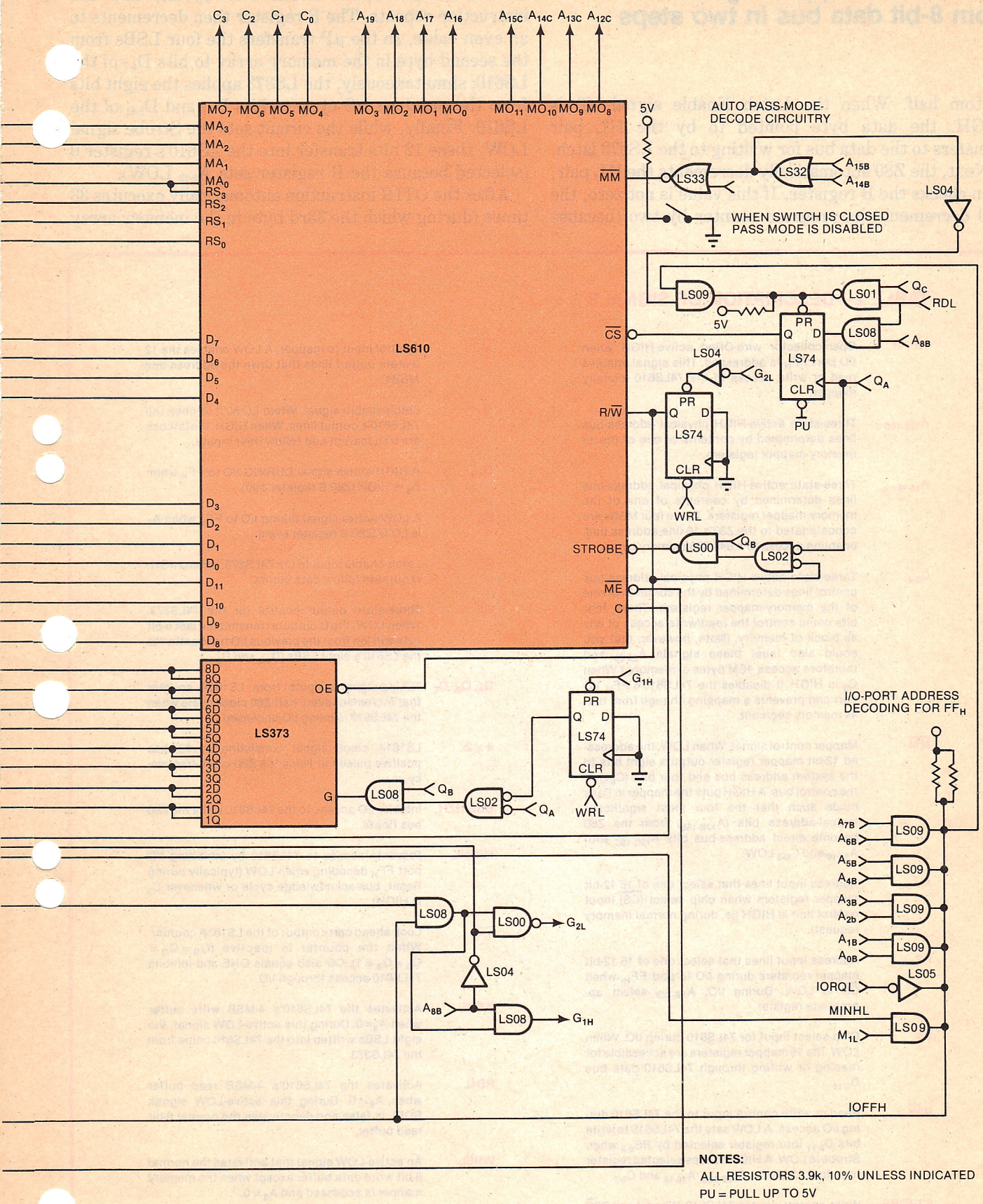


Fig 1—Built around a 74LS610 memory mapper *and SN74LS373 latch, this network extends the Z80 μP's address range to 1M bytes. Each of the 16 12-bit registers in the LS610 maps a 4k logical segment.*



NOTES:
ALL RESISTORS 3.9k, 10% UNLESS INDICATED
PU = PULL UP TO 5V

Feed 12-bit mapper registers from 8-bit data bus in two steps

bottom half. When the Latch Enable signal (G) is HIGH, the data byte pointed to by the HL pair transfers to the data bus for writing to the LS373 latch.

Next, the Z80 automatically decrements the HL pair, then checks the B register. If this value is not zero, the Z80 decrements its program counter by two (because the OTIR instruction needs two bytes), and the instruction repeats. The B register then decrements to an even value, so the μP transfers the four LSBs from the second byte in the memory array to bits D_{4-7} of th LS610; simultaneously, the LS373 applies the eight bits from the previous I/O cycle to bits D_{0-3} and D_{8-11} of the LS610. Finally, while the circuit sets the Strobe signal LOW, these 12 bits transfer into the LS610's register 0 (selected because the B register sets A_{9-12} LOW).

After the OTIR instruction automatically executes 33 times (during which the 33rd byte in the memory array

TABLE 2—DESCRIPTION OF SIGNALS

I0FFH	Open-collector wire-ORed active-HIGH when I/O port FF_H is addressed. This signal enables read or write access to the 74LS610 memory mapper.
$A_{12C-15C}$	Three-state active-HIGH physical address-bus lines determined by contents of one of the 16 memory-mapper registers.
A_{16-19}	Three-state active-HIGH physical address-bus lines determined by contents of one of the memory-mapper registers. These four MSBs are concatenated to the Z80's 16-line address bus, enabling access to 1,048,576 memory bytes.
C_{0-3}	Three-state active-HIGH physical address-bus control lines determined by the contents of one of the memory-mapper registers. These four bits could control the read/write access of any 4k block of memory. (Note, however, that you could also label these signals A_{20-23} and therefore access 16M bytes of memory.) When C_3 is HIGH, it disables the 74LS610's FF_H I/O port and prevents a mapping change from that 4k memory segment.
$\overline{MM}$	Mapper control signal. When LOW, the addressed 12-bit mapper register outputs eight bits to the system address bus and four bits (C_{0-3}) to the control bus. A HIGH puts the mapper in Pass mode such that the four most significant logical-address bits ($A_{12B-15B}$) from the Z80 become direct address-bus bits $A_{12C-15C}$ with A_{16-19} and C_{0-3} LOW.
MA_{0-3}	Address input lines that select one of 16 12-bit mapper registers when chip select ($\overline{CS}$) input control line is HIGH (ie, during normal memory request).
RS_{0-3}	Address input lines that select one of 16 12-bit mapper registers during I/O to port FF_H, when $\overline{CS}$ is LOW. During I/O, A_{9B-12B} select appropriate register.
$\overline{CS}$	Chip-select input for 74LS610 during I/O. When LOW, the 16 mapper registers are accessible for reading or writing through 74LS610 data bus D_{0-11}.
R/$\overline{W}$	Read or write control input to the 74LS610 during I/O access. A LOW sets the 74LS610 to write bits D_{0-11} into register selected by RS_{0-3} when Strobe is LOW. A HIGH causes selected register to output 12 bits to $A_{12C-15C}$, A_{16-19} and C_{0-3}.
STROBE	Write-control input to 74LS610. When it and R/$\overline{W}$ are LOW, it writes D_{0-11} into the mapper register selected by RS_{0-3}.
$\overline{ME}$	Control input to mapper. A LOW enables the 12 3-state output lines that drive the address-bus MSBs.
C	Latch-enable signal. When LOW, it latches the 74LS610's output lines. When HIGH, the latches are transparent and follow their inputs.
G_{1H}	A HIGH-active signal DURING I/O to FF_H when A_8 is HIGH (Z80 B register odd).
G_{2L}	A LOW-active signal during I/O to FF_H when A_8 is LOW (Z80 B register even).
G	Latch-enable input to the 74LS373. When HIGH, Q outputs follow data inputs.
OE	Three-state output control for the 74LS373. When LOW, the Q outputs transmit the last 8-bit byte written from the previous I/O transaction to the LS610's eight LSBs (D_{0-3} and D_{8-11}).
Q_A, Q_B, Q_C	Timing-signal outputs from LS161A counter that increment every half Z80 clock cycle when the 74LS610 is being I/O accessed.
Φ×2	LS161A clock signal consisting of 40-nsec positive pulses at twice the Z80 clock frequency.
BUS HZH	Inhibits I/O access to the 74LS610 while the Z80 bus floats.
MINHL	Prevents access to 74LS610 by inhibiting I/O port FF_H decoding when LOW (typically during Reset, bus-acknowledge cycle or whenever C_3 is HIGH)
CO	Look ahead carry output of the LS161A counter. When the counter is inactive ($Q_D = Q_C = Q_B = Q_A = 1$), CO also equals ONE and inhibits 74LS610 access through I/O.
WR4L	Activates the 74LS610's 4-MSB write buffer when $A_8 = 0$. During this active-LOW signal, the eight LSBs written into the 74LS610 come from the 74LS373.
RD4L	Activates the 74LS610's 4-MSB read buffer when $A_8 = 0$. During this active-LOW signal, RDEL is false and deactivates the normal 8-bit read buffer.
WREL	An active-LOW signal that activates the normal 8-bit write data buffer except when the memory mapper is accessed and $A_8 = 0$.
RDEL	An active-LOW signal that activates the normal 8-bit read data buffer except when the memory mapper is being accessed and $A_8 = 0$.

is ignored), B-register contents equal zero, and the instruction terminates. This 32-byte transfer to the LS610 takes 180 μsec for a 4-MHz Z80A and 120 μsec for a 6-MHz Z80B.

Table 1 shows the memory image of the 33-byte array during instruction execution—but only for the INIR (memory read) and OTIR (memory write) instructions during which the HL pair increments. You can read or write memory in the opposite direction with the INDR and OTDR instructions, for which the HL pair decrements.

An INIR read reverses the process

The discussion so far has detailed a memory write with the OTIR instruction; for an INIR read, the reverse process occurs. This time assume that you've set the C register to FF_H, the B register to 00100001 and the HL pair to point to the first byte in a 33-byte

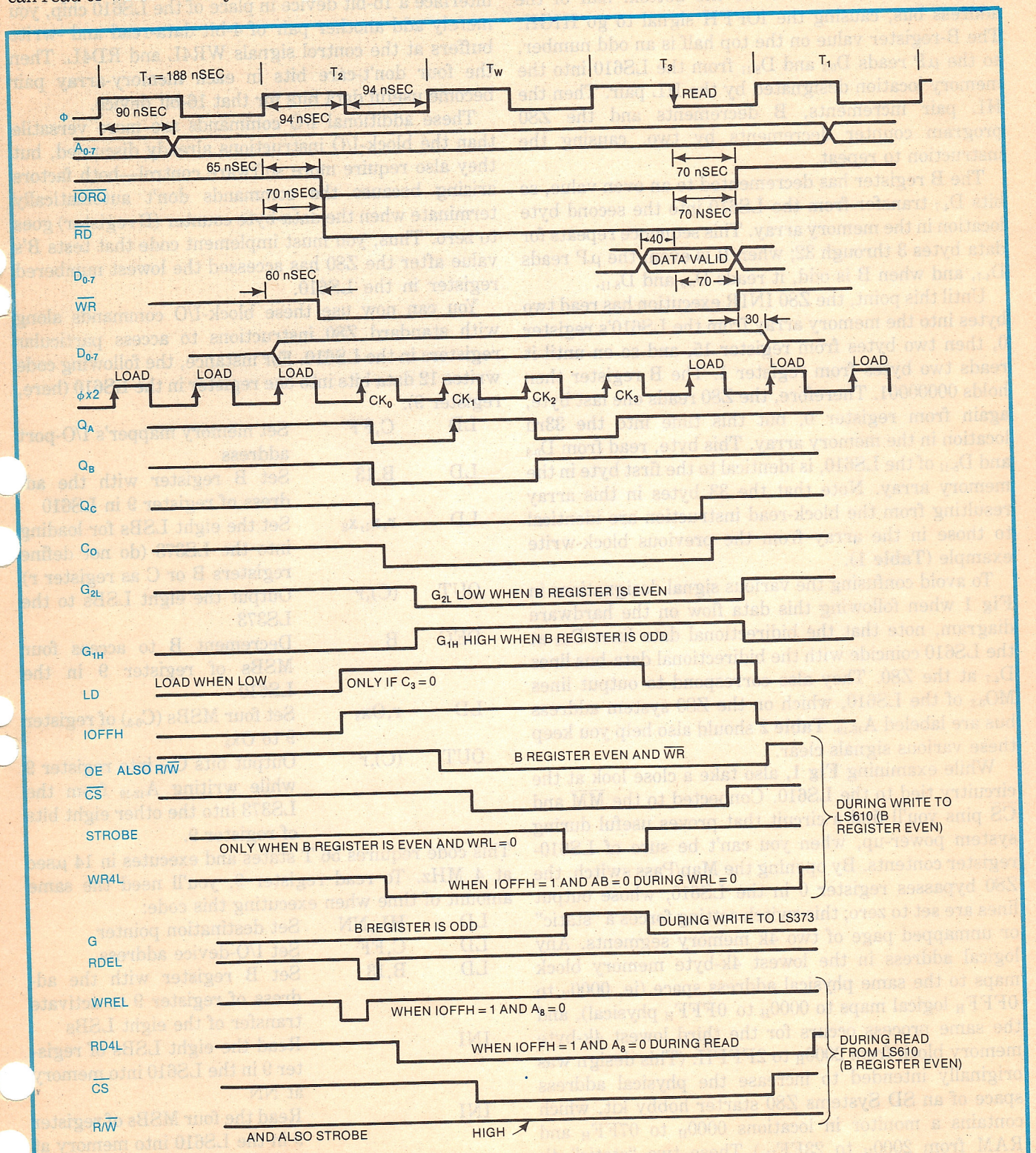


Fig 2—Shown here for a 6-MHz Z80B, *these timing relationships also hold for the 2-MHz Z80 and the 4-MHz Z80A.*

Transfer data to the mapper with block-I/O instructions

memory array.

When the Z80 encounters an INIR instruction, it performs the following operations. As before, C-register contents transfer to the bottom half of the address bus, causing the IOFFH signal to go HIGH. The B-register value on the top half is an odd number, so the μP reads D_{0-3} and D_{8-11} from the LS610 into the memory location designated by the HL pair. Then the HL pair increments, B decrements and the Z80 program counter decrements by two, causing the instruction to repeat.

The B register has decremented to an even value, so bits D_{4-7} transfer from the LS610 into the second byte location in the memory array. This sequence repeats for data bytes 3 through 32; when B is even, the μP reads D_{4-7}, and when B is odd, it reads D_{0-3} and D_{8-11}.

Until this point, the Z80 INIR execution has read two bytes into the memory array from the LS610's register 0, then two bytes from register 15, and so on until it reads two bytes from register 1. The B register then holds 00000001. Therefore, the Z80 reads one last byte, again from register 0, but this time into the 33rd location in the memory array. This byte, read from D_{0-3} and D_{8-11} of the LS610, is identical to the first byte in the memory array. Note that the 33 bytes in this array resulting from the block-read instruction are identical to those in the array from the previous block-write example (**Table 1**).

To avoid confusing the various signal designations in **Fig 1** when following this data flow on the hardware diagram, note that the bidirectional data lines D_{0-3} at the LS610 coincide with the bidirectional data-bus lines D_{4-7} at the Z80. They also correspond to output lines MO_{0-3} of the LS610, which on the Z80 system address bus are labeled A_{16-19}. **Table 2** should also help you keep these various signals clear.

While examining **Fig 1**, also take a close look at the circuitry tied to the LS610. Connected to the $\overline{MM}$ and $\overline{CS}$ pins you'll find a circuit that proves useful during system power-up, when you can't be sure of LS610-register contents. By opening the Map/Pass switch, the Z80 bypasses register 0 in the LS610, whose output lines are set to zero; this switch setting forces a "static" or unmapped page of two 4k memory segments. Any logical address in the lowest 4k-byte memory block maps to the same physical address space (ie, 0000_H to $0FFF_H$ logical maps to 0000_H to $0FFF_H$ physical), and the same process occurs for the third lowest 4k-byte memory block from 2000_H to 2FFFH. (This design was originally intended to increase the physical address space of an SD Systems Z80 starter hobby kit, which contains a monitor in locations 0000_H to $07FF_H$ and RAM from 2000_H to $23FF_H$.) These two "static" 4k segments disable LS610 registers 0 and 2 so that when you open the Map/Pass switch, you need merely set register B to 20_H to perform a read or 21_H for a write. A block transfer for 15 registers in this scheme takes 167 μsec at 4 MHz.

With the Map/Pass switch closed, the circuit's operation is modified by additional circuitry that prevents access to the LS610 unless the B register's two MSBs are zero, limiting system address space. To use the four bits reserved for control purposes to interface a 16-bit device in place of the LS610 chip, you merely add another pair of 4-bit data-read and -write buffers at the control signals WR4L and RD4L. Then the four don't-care bits in each memory-array pair become useful data bits for that 16-bit device.

These additional I/O commands are more versatile than the block-I/O instructions already discussed, but they also require more software control—both factors arising because the commands don't automatically terminate when the data-byte counter (B register) goes to zero. Thus, you must implement code that tests B's value after the Z80 has accessed the lowest numbered register in the LS610.

You can now use these block-I/O commands along with standard Z80 instructions to access particular registers in the LS610. For instance, the following code writes 12 data bits into one register in the LS610 (here, register 9):

LD	C,FF	Set memory mapper's I/O-port address
LD	B,13	Set B register with the address of register 9 in LS610
LD	r,x_1,x_2	Set the eight LSBs for loading into the LS373 (do not define registers B or C as register r)
OUT	(C),r	Output the eight LSBs to the LS373
DEC	B	Decrement B to access four MSBs of register 9 in the LS610
LD	r,Ox_3	Set four MSBs (C_{0-3}) of register 9 to Ox_3
OUT	(C),r	Output bits C_{0-3} into register 9 while writing A_{12-19} from the LS373 into the other eight bits of register 9.

This code requires 56 T states and executes in 14 μsec at 4 MHz. To read register 9, you'll need the same amount of time when executing this code:

LD	HL,NN	Set destination pointer
LD	C,FF	Set I/O-device address
LD	B,13	Set B register with the address of register 9 to activate transfer of the eight LSBs
INI		Read the eight LSBs of register 9 in the LS610 into memory at NN
INI		Read the four MSBs of register 9 in the LS610 into memory at NN+1.

A final example writes six data bytes into three adjacent LS610 registers (here, registers 14, 13 and 12)

Watch mapping-register software carefully

without modifying any other registers:

LD	HL,NN	Set memory-source pointer
LD	C,FF	Set memory mapper's I/O-port address
LD	B,1E	Set B to activate hardware for the eight LSBs to transfer to the LS373
OUTI		Write eight bits into LS373
OUTI		Write four bits (C_{0-3}) directly into register 14 while the LS373 writes A_{12-19} into the eight LSBs of register 14
OUTI		Write eight bits to LS373
OUTI		Write 12 bits to register 13 in LS610
OUTI		Write eight bits to LS373
OUTI		Write 12 bits to register 12 of LS610.

This code takes 120 T states, equivalent to 30 μsec at 4 MHz. Note that you can access sections of **Table 1**'s memory array to set up this write sequence.

In these or any of the indexed memory-mapping schemes described here, the software that modifies the contents of one of the 16 LS610 mapping registers should not execute from the physical address space determined by that same mapping register. Note, too, that the Z80 can recognize interrupts during block I/O, but the associated interrupt-service routines should not change LS610 index-register values that control their own execution address space. **EDN**

References

1. LeMair, Ian, "Indexed Mapping Extends Microprocessor Addressing Range," *Computer Design,* August, 1980.
2. Ogdin, Carol Anne, "An innovative TTL chip gives 8-bit μPs new life," *EDN,* November 5, 1980, pgs 269-273.
3. *Mostek Z80 Microcomputer Software Programming Guide,* MK78515, Mostek Corp, Carrollton, TX.

Author's biography

John Tomaszewski is a systems consultant in Cross Plains, WI. Formerly employed at National Electrostatics Corp (Middleton, WI), he earned a BSEE degree from the University of Wisconsin. John lists home computing as a primary spare-time activity.

Article Interest Quotient (Circle One)
High **488** ***Medium*** **489** ***Low*** **490**