The Z-80 in Parallel

Bob Loewer
Micro Diversions Inc
7900 Westpark Dr
Suite 308
McLean VA 22101

Many design engineers have introduced various types of parallel processing into systems in order to achieve higher throughput rates. Almost without exception though, these applications have been limited to medium and large scale computers due to price and complexity.

In the past two years, microprocessors have reached a level of sophistication which makes them candidates for parallel processing systems. Such systems could conceivably offer minicomputer performance at microcomputer cost. This article is an investigation of that idea.

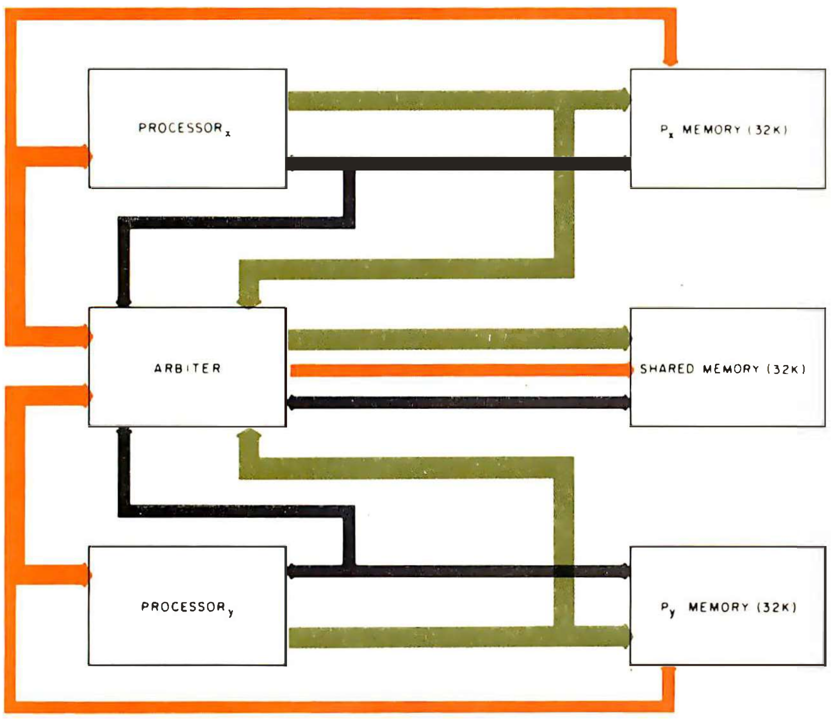


Figure 1: The author's parallel Z-80 system. Both processors work independently, each supported by 32 K bytes of programmable memory. The processors are linked by 32 K bytes of shared programmable memory. The shared memory, addressable by either processor as the upper 32 K, has its own address and data buses. Shared memory conflicts are resolved by the arbiter circuit shown in figure 2a.

The Z-80

The Z-80 microprocessor, manufactured by Zilog, is a third generation LSI device which offers full software compatibility with the 8080 processor. Upgraded features provided by the Z-80 include: two sets of exchangeable registers, indexing, a full range instruction set (including register or memory bit operations), eleven addressing modes, a nonmaskable interrupt, dynamic memory refresh address generation, and an interrupt register to provide a high speed vectored interrupt response to any location in memory.

The Z-80's minimum number of control bus signals makes it easy to interface in multiple processor configurations.

System Layout

My design consists largely of two Z-80 microprocessors (processor X and processor Y) operating independently, each supported by 32 K bytes of programmable memory (see figure 1). The processors are indirectly linked by 32 K bytes of common memory, making a system total of 96 K bytes. The shared memory, addressable by either processor as the upper 32 K, has its own address and data buses. Data or address signals are gated onto their respective bus when (1) either processor performs an operation involving a read or write against the shared memory, or (2) either processor attempts an op code fetch from the shared memory, or (3) machine instructions combine (1) and (2).

Shared memory bus conflicts are resolved by the arbiter (see figure 2a). Since the processors use opposite phases of the clock, requests for bus access can never be initiated at exactly the same time. However, depending upon the instruction sequences being executed, bus request conflicts can occur. This problem, summarized in table 3, has been carefully examined and is represented by figure 3b. It illustrates what is assumed to

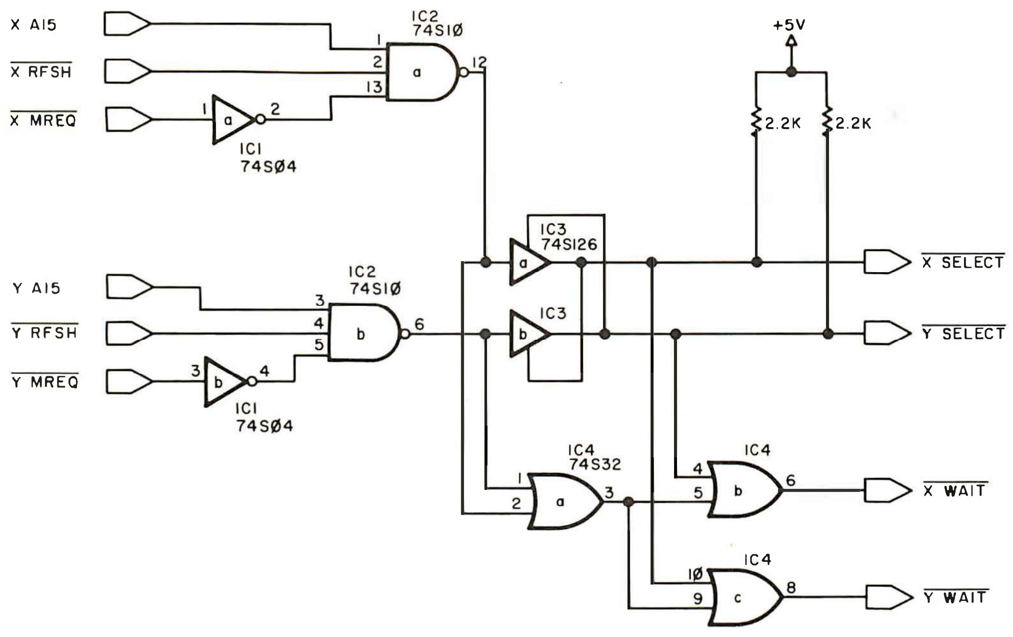


Figure 2a: The shared memory arbiter. This circuit resolves conflicts between the two processors if both attempt to gain simultaneous access to the shared memory bus. For example, a request from processor X ($\overline{XMREQ}$ low) will cause IC3a to drive the $\overline{XSELECT}$ line low and will also disable IC3b. Processor Y will be locked out during X's memory request. If Y makes a memory request while locked out, the output of IC4a will go low, activating the $\overline{YWAIT}$ line.

About the Author

Bob Loewer is an employee of the Telenet Communications Corporation and a graduate student at the University of Maryland at College Park. He is cofounder of Micro Diversions Inc, a company involved with microcomputers and microcomputer education. This article describes Bob's early research on parallel microprocessor systems.

be the worst possible case of bus conflict: both processors simultaneously executing shared memory read or write instructions from the shared memory. Of course, one cannot predict when each processor will attempt to access the shared memory, so all possible interprocessor state relationships have been investigated.

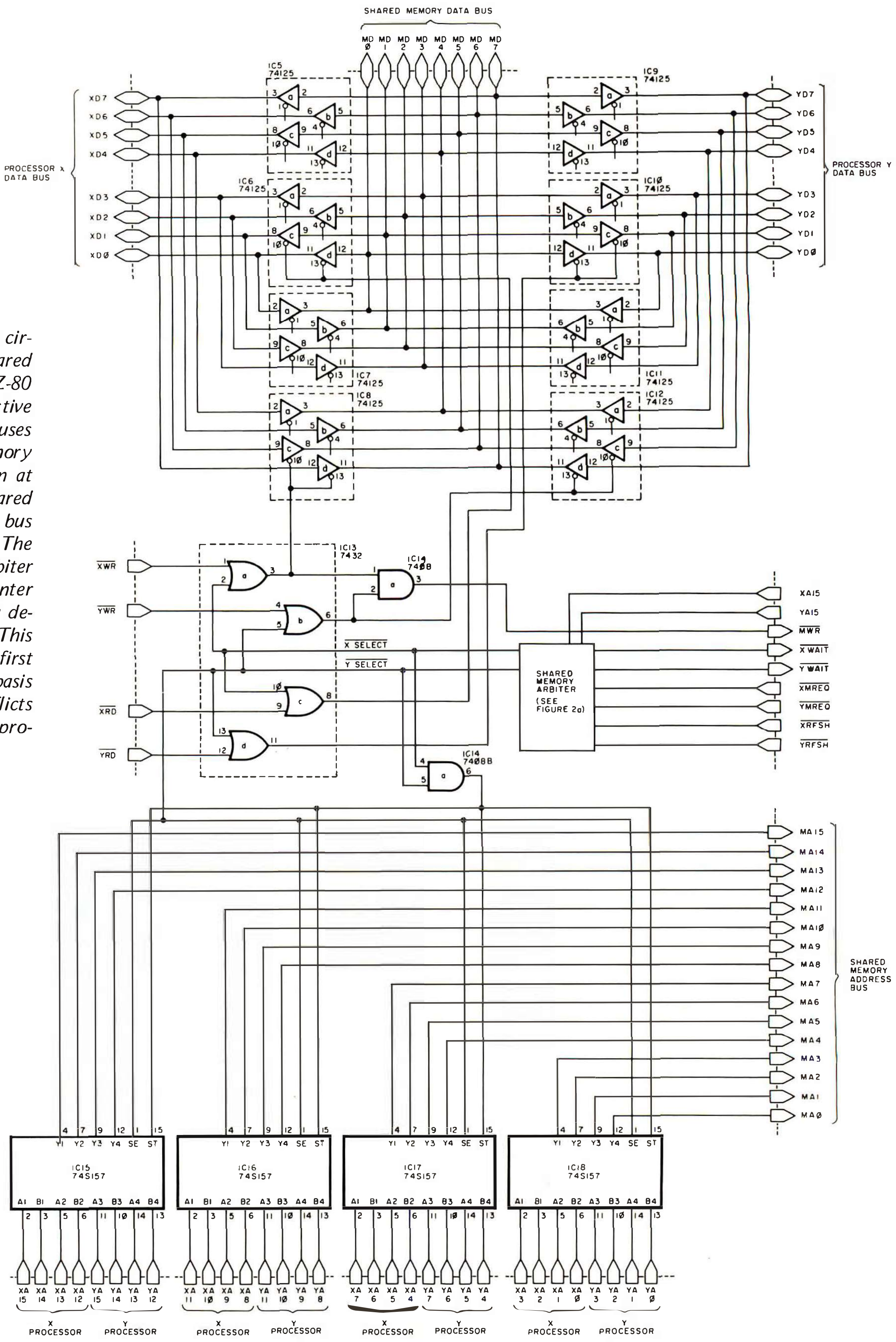
The basic memory read or write instruction has seven "T" cycles (T is defined as the duration of one clock period). The T states and their functions are:

M1,T1		Op code fetch
M1,T2		Op code fetch
M1,T3	Instruction decoding	Op code fetch
M1,T4	Instruction decoding	Op code fetch
M2,T1	Memory read or write operation	
M2,T2	Memory read or write operation	
M2,T3	Memory read or write operation	

The M cycles are machine cycles. Table 3 shows the seven interprocessor T state alignments: M1,T1 active for one processor when states M1,T1 thru M2,T3 for the other are active. Figure 3b illustrates an example of the processor request signals and signals from the conflict arbitration logic. Note that after a very short period (maximum of seven clock cycles) the arbiter synchronizes and thereby provides complete cooperation between the two processors' fetch and execution cycles by putting one of the processors into one or two wait states. Further, in the seven possible interprocessor T state relationships, there are two in which opposing shared memory access request signals are synchronized, in which case the arbiter does nothing. This means that, regardless of the processors' instruction

Number	Type	+5 V	GND
IC1	74S04	14	7
IC2	74S10	14	7
IC3	74S126	14	7
IC4	74S32	14	7
IC5	74125	14	7
IC6	74125	14	7
IC7	74125	14	7
IC8	74125	14	7
IC9	74125	14	7
IC10	74125	14	7
IC11	74125	14	7
IC12	74125	14	7
IC13	7432	14	7
IC14	7408	14	7
IC15	74S157	16	8
IC16	74S157	16	8
IC17	74S157	16	8
IC18	74S157	16	8

Table 1: Power wiring table for figures 2a and 2b.



SHARED MEMORY DATA BUS
MD Ø MD 1 MD 2 MD 3 MD 4 MD 5 MD 6 MD 7
IC5 74125
IC6 74125
IC7 74125
IC8 74125
IC9 74125
IC1Ø 74125
IC11 74125
IC12 74125
XD7 XD6 XD5 XD4 XD3 XD2 XD1 XDØ
PROCESSOR X DATA BUS
YD7 YD6 YD5 YD4 YD3 YD2 YD1 YDØ
PROCESSOR Y DATA BUS
IC13 7432
IC14 74Ø8
IC14 74Ø8B
XWR
YWR
XRD
YRD
X SELECT
Y SELECT
SHARED MEMORY ARBITER (SEE FIGURE 2a)
XA15 YA15 MWR X WAIT Y WAIT XMREQ YMREQ XRFSH YRFSH
MA15 MA14 MA13 MA12 MA11 MA1Ø MA9 MA8 MA7 MA6 MA5 MA4 MA3 MA2 MA1 MAØ
SHARED MEMORY ADDRESS BUS
IC15 74S157
IC16 74S157
IC17 74S157
IC18 74S157
X PROCESSOR
Y PROCESSOR

Figure 2b: Control circuitry for the shared memory parallel Z-80 system. The respective processor data buses and the shared memory data bus are shown at the top. The shared memory address bus is at the right. The shared memory arbiter is shown in the center (see figure 2a for a detailed schematic). This circuit works on a first come, first served basis to resolve all conflicts between the two processors.

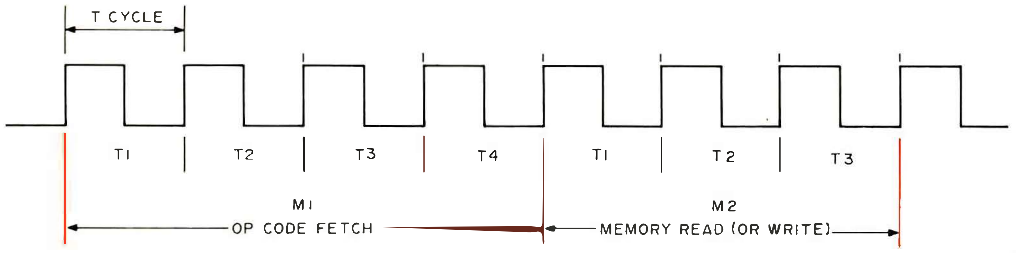


Figure 3a: The basic Z-80 memory read or write cycle. Clock periods are referred to as T cycles and the basic operations are referred to as M (for machine) cycles. The first machine cycle of any instruction is a fetch cycle (M1). Subsequent M cycles move data to or from the memory.

sequences, 86 percent of the time the system is at most one wait state away from synchronization. Thereafter, both processors can execute read and write instructions from the shared memory at 100 percent processor utilization, assuming the instruction synchronization is not lost.

Certainly opposing software will not consist solely of instructions which offer no bus interference. But it is clear that the most efficient method of solving the shared memory bus conflict problem is the one that will achieve short term interprocessor synchronization whenever possible.

Arbitration Logic

Each processor provides signals to the arbiter which identify a valid shared memory access request. IC2a and IC2b receive $\overline{\text{RFSH}}$, $\overline{\text{MREQ}}$, and A15 (the high order address bit signal) from their respective processors. $\overline{\text{MREQ}}$ indicates that a memory read or write operation is underway: either A15 line going high identifies the shared memory as the object of the request; and the $\overline{\text{RFSH}}$ lines insure that the dynamic memory refresh strobe from one processor will not interfere with the shared memory access request of the other.

IC3a and IC3b provide an opposing grant or deny shared memory bus access proviso that is strictly first come, first served. A request from, say, processor X will cause IC3a to drive $\overline{\text{XSELECT}}$ low, and coincidentally disable IC3b. Processor Y will be locked out for the length of processor X's memory request. Now suppose processor Y *does* make a request for bus access when processor X is using the bus. This condition will force IC4a to its low state, activating the $\overline{\text{YWAIT}}$ line. The wait signal will continue until processor X concludes its memory access. Under no circumstances, however, will processor Y be forced into more than one wait state for this processor X access. When $\overline{\text{XMREQ}}$ goes high, $\overline{\text{XSELECT}}$ follows

Continued on page 174

Beginning Event	Finishing Event	Stipulation	Delay Before Occurrence (ns)
XA15 high $\overline{\text{XRFSH}}$ high $\overline{\text{XMREQ}}$ low	to $\overline{\text{XSELECT}}$ low	$\overline{\text{YSELECT}}$ high	28
XA15 high $\overline{\text{XRFSH}}$ high $\overline{\text{XMREQ}}$ low	to $\overline{\text{XSELECT}}$ low	$\overline{\text{YSELECT}}$ low	25 after $\overline{\text{YSELECT}}$ goes high
YA15 high $\overline{\text{YRFSH}}$ high $\overline{\text{YMREQ}}$ low	to $\overline{\text{YSELECT}}$ low	$\overline{\text{XSELECT}}$ high	28
YA15 high $\overline{\text{YRFSH}}$ high $\overline{\text{YMREQ}}$ low	to $\overline{\text{YSELECT}}$ low	$\overline{\text{XSELECT}}$ low	25 after $\overline{\text{XSELECT}}$ goes high
XA15 high $\overline{\text{XRFSH}}$ high $\overline{\text{XMREQ}}$ low	to $\overline{\text{XWAIT}}$ low	$\overline{\text{YSELECT}}$ low	53
YA15 high $\overline{\text{YRFSH}}$ high $\overline{\text{YMREQ}}$ low	To $\overline{\text{YWAIT}}$ low	$\overline{\text{XSELECT}}$ low	53
$\overline{\text{XSELECT}}$ high	to $\overline{\text{YWAIT}}$ high	$\overline{\text{YSELECT}}$ low	22
$\overline{\text{YSELECT}}$ high	to $\overline{\text{YWAIT}}$ high	$\overline{\text{XSELECT}}$ low	22

Table 2: Timing considerations in the arbiter circuitry. The arbiter takes a finite amount of time for its logic circuits to effect the changes shown. The corresponding delays are shown at the right.

Continued from page 63

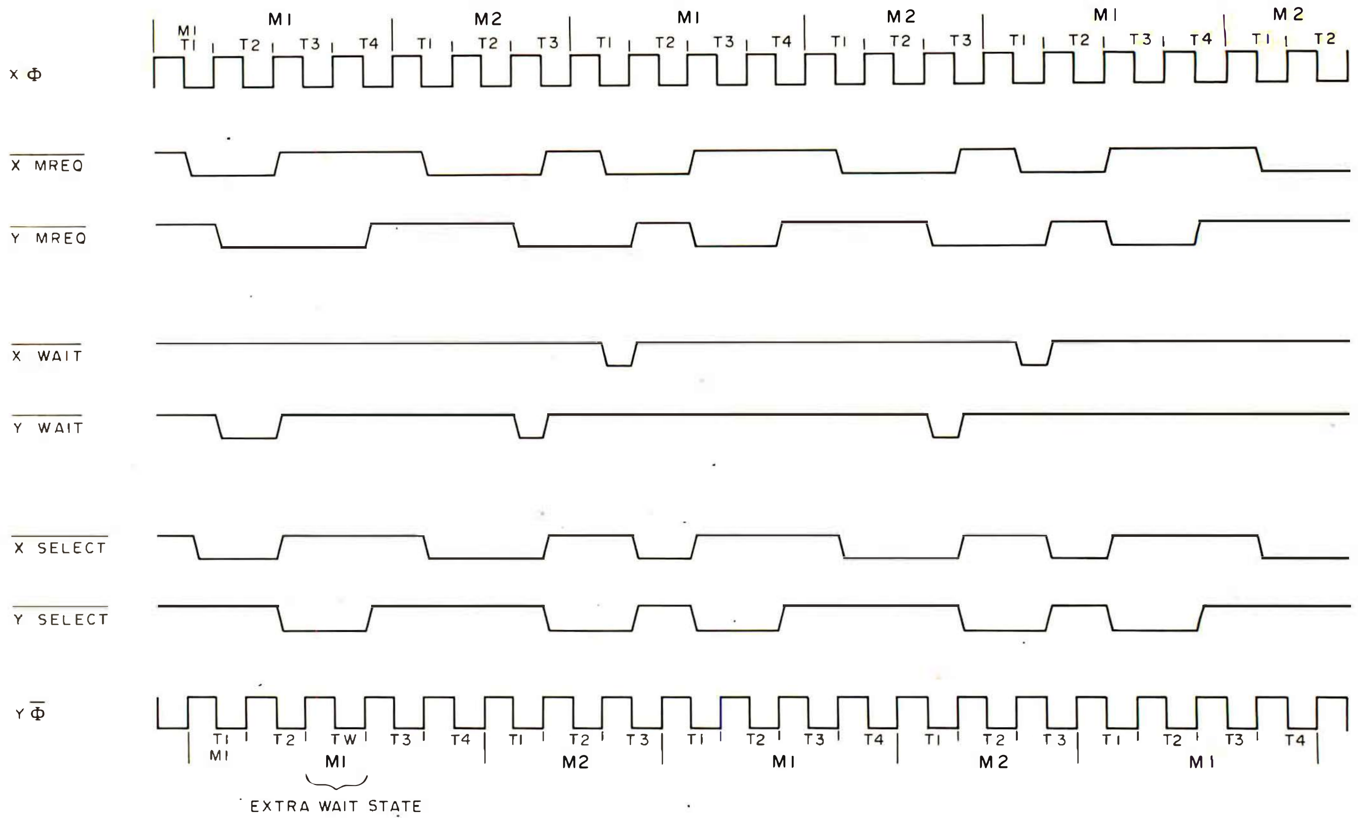


Figure 3b: M1,T1/M1,T1, one of the seven interprocessor T state alignments shown in table 1. Synchronization is achieved after three clock cycles in this example. A complete discussion of the M and T states can be found in the Z-80 technical manual. (Note that, even though the arbiter activates the wait lines at periods during synchronization, this has no effect on the processors because the lines are not sampled until the falling edge of T2.)

Inter-processor State Relationship P_X/P_Y	Synchronization Pattern M1,T1/M1,T3	M1,T1/M2,T3	Wait States P_X	P_Y	T States Until Synchronization
M1,T1/M1,T1		√	0	1	3
M1,T1/M1,T2	√		1	0	3
M1,T1/M1,T3	√		0	0	0
M1,T1/M1,T4	√		0	1	6
M1,T1/M2,T1	√		0	2	7
M1,T1/M2,T2		√	1	0	3
M1,T1/M2,T3		√	0	0	0

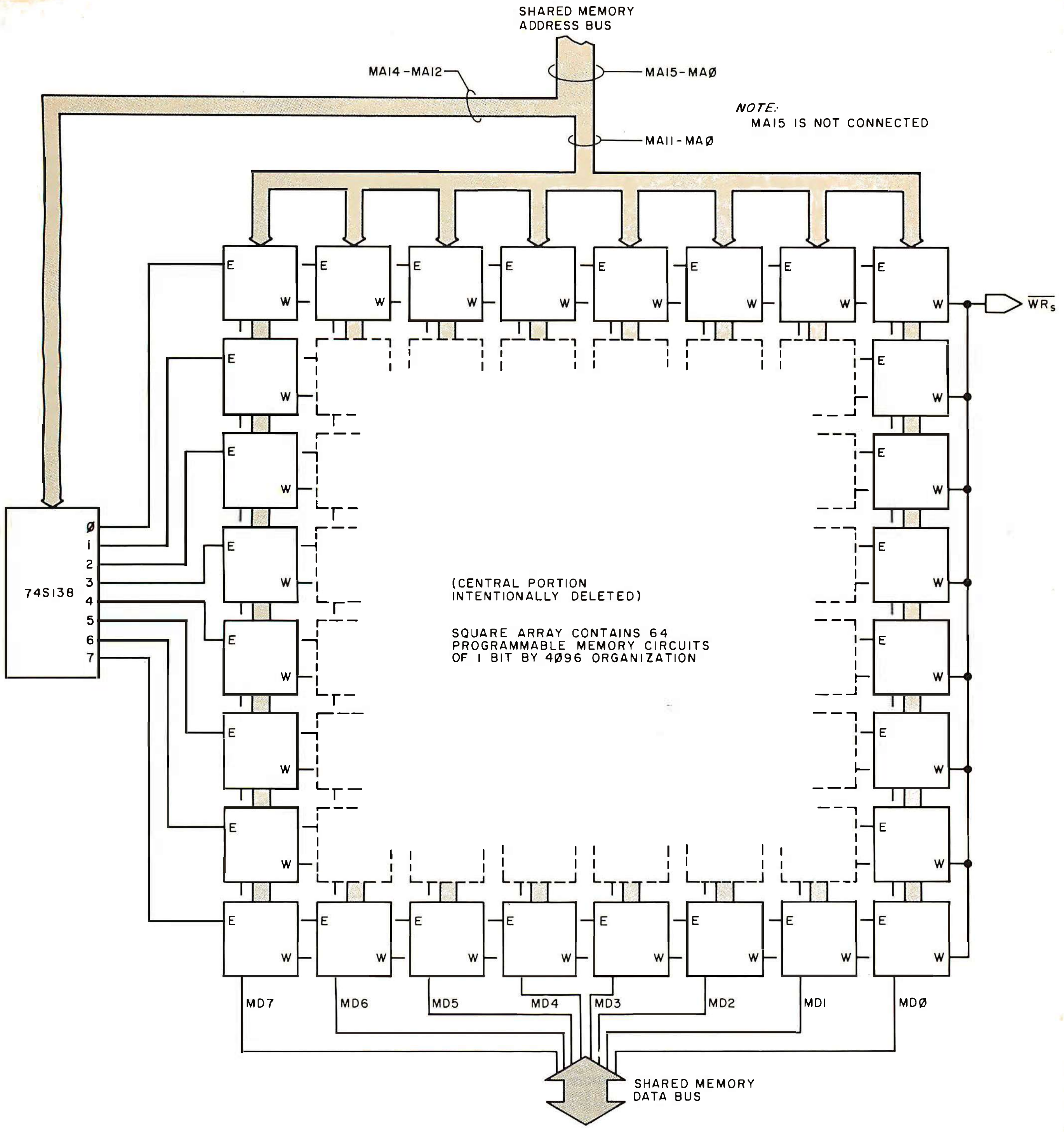
Table 3: Timing analysis for two Z-80 processors in parallel. The seven possible interprocessor state relationships are shown at left. The center column lists the two possible classes of operation for the parallel processors: the first (M1,T1/M1,T3) occurs when both processors are performing an op code fetch at the time of synchronization; the second is the case when one processor is performing an op code fetch while the other is performing a memory read or write. The wait states column indicates how many wait states each processor must undergo until synchronization occurs. Note that in two of the cases, the shared memory arbiter (see figure 2a) need not be employed, since the two processors fall into synchronization spontaneously.

suit, enabling IC3b; ie: granting processor Y's request.

System Timing

In order to choose a memory that is subject to overlapping access requests from more than one processor, system timing must be carefully examined. Important considerations in this design include the control logic propagation delay and the "window size" provided by the processors for read or write accesses to the common memory.

In the single processor system, the smallest memory access window of any Z-80 instruction occurs during the op code fetch cycle. The effective length of that cycle is a few nanoseconds less than 1.5 clock cycles (1.5 Φ). However, the dual processor configuration reduces the window size for two reasons: (1) the delay in processor selection (ie: the data gating signal) incurred by the control logic and, (2) the overlap of the memory request signals from opposing processors that is required to permit full speed operation by the processors. Further, the memory cycle time requirement becomes more stringent, accommodating from more than two clock cycles in a 1 processor system to less than one clock cycle in this system.



SHARED MEMORY ADDRESS BUS
MA14-MA12
MA15-MAØ
NOTE: MA15 IS NOT CONNECTED
MA11-MAØ
74S138
Ø
1
2
3
4
5
6
7
E
W
$\overline{WR_s}$
(CENTRAL PORTION INTENTIONALLY DELETED)
SQUARE ARRAY CONTAINS 64 PROGRAMMABLE MEMORY CIRCUITS OF 1 BIT BY 4Ø96 ORGANIZATION
MD7
MD6
MD5
MD4
MD3
MD2
MD1
MDØ
SHARED MEMORY DATA BUS

Figure 4: Block diagram of the shared memory. The memory is arranged in a square array of 64 static programmable memory integrated circuits with 4096 bits per circuit.

The clock that drives the processors operates at 2.5 MHz, defining a basic cycle time of 400 ns. With this information it is now possible to calculate the operating characteristics required of the shared memory.

As stated earlier, the memory access window depends on the control logic switching delay and the request signals overlap. It has been shown (see figure 3) that the smallest window occurs at times of bus request conflicts, and that this window has a length of one clock cycle. The equation, then, for actual window length is:

$$L_a = \Phi - \delta - T_c - T_d$$

where:

Φ	=	400 ns (1 clock cycle)
δ	=	maximum delay in falling edge of $\overline{\text{MREQ}}$
T_c	=	maximum propagation delay of control logic
T_d	=	maximum propagation delay of decoding logic

Examining the control logic timing shown in table 2, we observe that the maximum timing delays from either $\overline{MREQ}$ line going active until the arbiter sets the corresponding $\overline{SELECT}$ line active are as follows:

IC In Signal Path	Maximum Propagation Delay (ns)
74S04	5
74S10	5
74LS126	+18
Total Arbiter Delay:	28 ns

Substituting into the equation for L_a, we obtain:

$$\begin{aligned} L_a &= \Phi - \delta - T_c - T_d \\ &= 400 - 20 - 28 - T_d \\ &= 352 - T_d \text{ ns} \end{aligned}$$

The switching delay of the decoding logic, T_d ($\overline{SELECT}$ active until the memory receives the signals), further reduces the memory access window. Referring to figures 2b and 4, the signal path T_d is:

$$\begin{aligned} T_d &= 7408 + \max\,[74S157 \text{ (select)}, \\ &\quad 74S157 \text{ (enable)}] + 74S138 \\ &= 19 + \max\,[15, 11] + 12 \\ &= 46 \text{ ns} \end{aligned}$$

Finally,

$$\begin{aligned} L_a &= 352 - T_d \\ &= 306 \text{ ns} \end{aligned}$$

This allows plenty of time for a memory access operation; so much time, in fact, that we do not need the faster and more expensive bipolar programmable memory.

We must also consider the memory cycle timing (L_c), reduced by this two processor system to $\Phi - \delta$:

$$\begin{aligned} L_c &= \Phi - \delta \\ &= 400 - 20 \\ &= 380 \text{ ns} \end{aligned}$$

It is good design practice to calculate delays in the system using the maximum time figures rather than typical ones, and to adjust the results by including a safety margin. Accordingly, we specify the following requirements for the shared static programmable memory:

- Access time 280 ns or less
- Cycle time 350 ns or less

Conclusions and Possible Applications

The principle advantage of two (or more) parallel processors performing complementary tasks is the cost savings. For example, let us say that we operate a packet switching network in which multiple microprocessors perform the relay functions of each node, such as the TELENET of Telenet Communications Corporation. Our responsibilities include insuring data reliability (eg: using cyclic redundance coding (CRC), checksum, etc), doing format checks, and recognizing the destination of the traffic and routing it to another node in the network. Further, this service must be provided at high speed.

Clearly, for one processor to perform these and other nodal functions without some delay, high performance and high cost systems would be required. Conversely, multiple microprocessors could perform all of these tasks in parallel at a significant reduction in cost.

For the experimenter, a multiprocessor system doesn't appear to offer much beyond an interesting diversion in design engineering. As mentioned earlier, the benefit from this type of design is increased throughput (by virtue of the reduced per unit cost). This is an idea that has little significance for persons with a dedicated system.

One possible application does come to mind, however. Many 8080 system owners are upgrading to the Z-80 for the expanded instruction set, but for some, direct replacement of a processor board is not possible. Why not consider adding a Z-80 with your current system acting as a front end? Admittedly, it seems like a bit of overkill, but it is an inexpensive way ($8 for the interface circuitry of this design) to upgrade. Of course, after installing another processor, the owner must write an operating system to accommodate the addition; but that's part of the continuing challenge to be found in the world of microprocessors.■

REFERENCES

Z80-CPU Technical Manual, Zilog, 170 State St, Los Altos CA 94022, 1976.