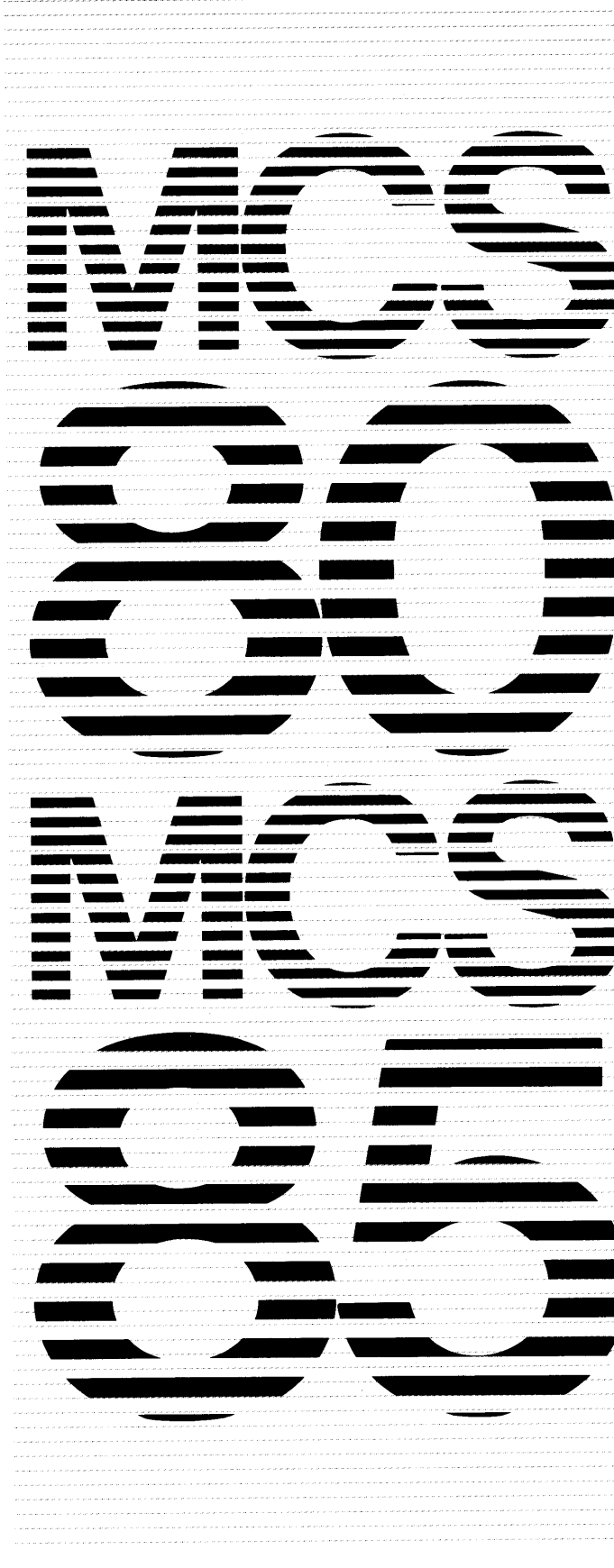
Appendix
Applications
of MCS-85™



Appendix
Applications of MCS-85™

Contents

APPENDIX 1
APPLICATIONS OF MCS-85™

SECTION 1
INTRODUCTION TO MCS-85™ APPLICATIONS

When the first microprocessor was introduced about five years ago, it was largely ignored by the electronics industry. However, since that inasupicious beginning, this new device has become the hottest topic in current technology. As more and more product designers become familiar with the capabilities of microcomputers, the number of new applications increases geometrically. In most of these applications, the new technology has been used to replace designs which were formerly implemented with TTL logic and under-utilized minicomputers. However, an increasing number of products are surfacing which would have been impractical prior to the microcomputer era.

Microcomputers are being applied to a wide range of data communications tasks. The field of telephone equipment is being invaded by systems which control and monitor calls. Point of sale terminals are increasing daily with the addition of interface to coin changers, electronic scales and remote computers. Small stand-alone computers are relying heavily upon microcomputers in teleprocessing, timesharing, data base management and similar interactive applications. An increasing number of microcomputer-based data terminals are providing local interactive intelligence with programmable character sets, vector generation and the pre-processing of data.

Instrumentation is widely utilizing the microprocessor for a variety of control and arithmetic processing functions. Microcomputers are controlling laboratory equipment such as oscilloscopes, DVM's, network analyzers and frequency synthesizers. Medical electronics are crediting microcomputers with tasks such as patient monitoring, blood analysis and X-ray scanning. Travel is becoming microcomputerized by automotive control, air and ocean navigation equipment and rapid transit systems.

MCS-85™ SYSTEM

Many possible microcomputer applications have been overlooked because of the design tasks required to build the microcomputer. These tasks include the system clock, read/write memory, I/O ports, serial communications interface and bus control logic. The MCS-85 system will enable the design engineer to concentrate on the application of the microcomputer, rather than on the implementation details.

The MCS-85 is yet another family of components which has the potential to provide a solution to the three problems which will always plague designers: cost, size and power. The reduced component count of an MCS-85 microcomputer, coupled with the increased integration of functions reduces both cost and size while increasing power.

Sample Applications

Calculating Oscilloscope
Blood Analyzer
Programmable Video Game
Process Control System
Line Printer
Intelligent Terminal
N.C. Machine
Digital Multimeter
Graphic Terminal
Automotive Control
Navigation Equipment
Vending Machine
Spectrum Analyzer
Front End Processor
Credit Verifier
Disk Controller
Patient Monitor
Network Analyzer
Frequency Synthesizer

APPLICATION	PERIPHERAL DEVICES ENCOUNTERED	MCS-85™ COMPONENTS	
Intelligent Terminals	Cathode Ray Tube Display	8275	8085A
	Printing Units	8155	8355
	Synchronous and Asynchronous data lines	8251	
	Cassette Tape Unit		
	Keyboards	8279	
Gaming Machines	Keyboards, pushbuttons and switches	8279	8085A
	Various display devices		8355
	Coin acceptors	8155	
	Coin dispensers		
Cash Registers	Keyboard or Input Switch Array	8279	8085A
	Change Dispenser	8155	8355
	Digital Display		
	Ticket Printer		
	Magnetic Card reader		
	Communication interface	8273	
Accounting and Billing Machines	Keyboard	8279	8085A
	Printer Unit	8155	8355
	Cassette or other magnetic tape unit	8257	
	"Floppy" disks	8271	
Telephone Switching Control	Telephone Line Scanner	8253	8085A
	Analog Switching Network		8355
	Dial Registers	8155	
	Class of Service Parcel		
Numerically Controlled Machines	Magnetic or Paper Tape Reader	8155	8085A
	Stepper Motors		8355
	Optical Shaft Encoders		
Process Control	Analog-to-Digital Converters	8155	8085A
	Digital-to-Analog Converters		8355
	Control Switches	8279	
	Displays		

Baud Rate Generator

Shown in Figure 2 is a minimum system configuration with the 8156 timer output connected to an 8085 interrupt input.

This configuration allows convenient use of the timer as a baud rate generator. A 6.144 MHz crystal is used as the frequency control element of the 8085A, providing integral divisors for the standard baud rates (300, 600, 1200, 2400, 4800, 9600 baud). The timer is programmed with the appropriate divisor (Figure 1) for the selected baud rate resulting in one pulse on the timer output for each bit cell time. The clock output (CLK) of the 8085A is used to clock the timer (TIMERIN). The frequency of this clock is one-half the crystal frequency or in this example 3.072 MHz. $\overline{\text{TIMEROUT}}$ now provides a crystal controlled pulse train at the baud rate selected.

Serial Communications

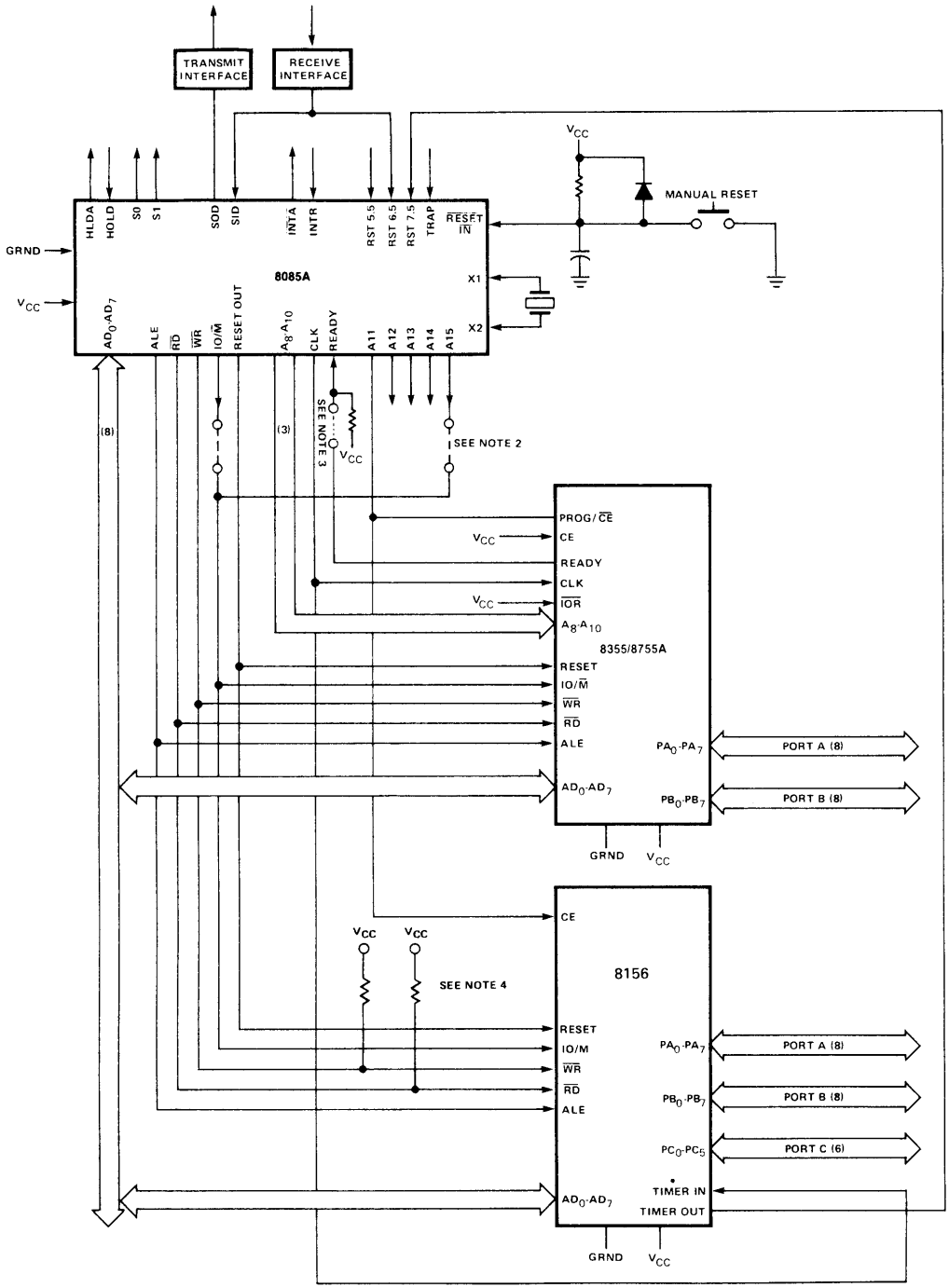
By feeding the $\overline{\text{TIMEROUT}}$ signal of the 8156 back to the edge triggered RST 7.5 input of the 8085A, the processor can be interrupt driven at the required baud rate. As shown in Figure 1, the minimum system supports serial communications with only the addition of the send and receive interface circuits.

The SID (SERIAL INPUT DATA) line and the SOD (SERIAL OUTPUT DATA) line are connected directly to a TTY or RS232 interface circuit. Assuming inverted data at the SID input, a direct connection is made to the RST6.5 input for detection of the start bit.

Additional insight into using the 8085's serial I/O lines in communications application can be found in Section 2 of this Appendix.

BAUD RATE	COUNT (DECIMAL)
300	10,240
600	5,120
1200	2,560
2400	1,280
4800	640
9600	320

FIGURE 1. BAUD RATES



NOTE 1: TRAP, INTR, AND HOLD MUST BE GROUNDED IF THEY AREN'T USED.
NOTE 2: USE IO/M FOR STANDARD I/O MAPPING. USE A15 FOR MEMORY MAPPED I/O.
NOTE 3. CONNECTION IS NECESSARY ONLY IF ONE T_{WAIT} STATE IS DESIRED.
NOTE 4: PULL-UP RESISTORS RECOMMENDED TO AVOID SPURIOUS SELECTION WHEN $\overline{RD}$ AND $\overline{WR}$ ARE 3-STATED.

FIGURE 2. MINIMUM SYSTEM CONFIGURATION

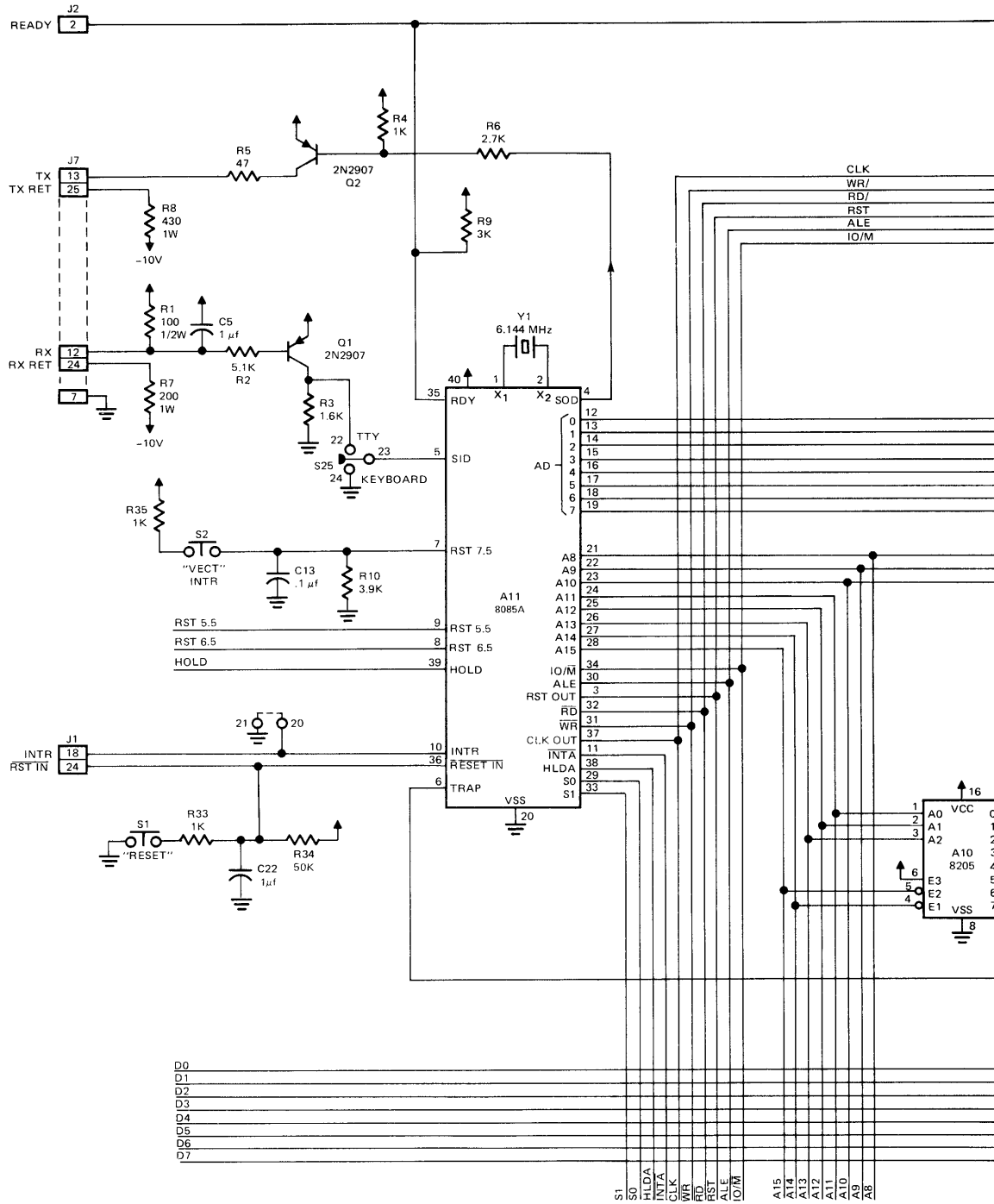
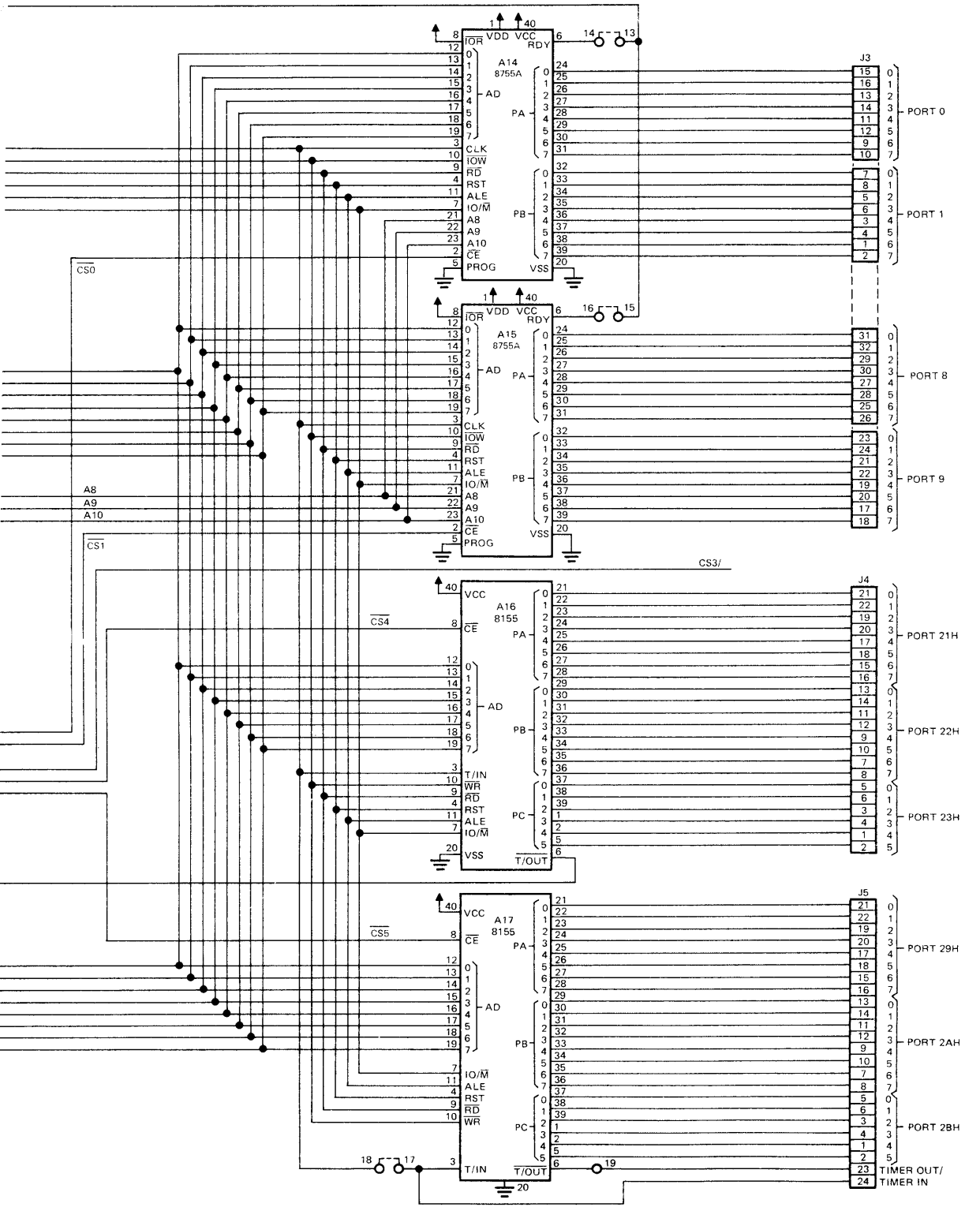


FIGURE 3. SMALL SYSTEM SCHEMATIC (similar to the schematic of Intel's SDK-85)


A14
8755A
A15
8755A
A16
8155
A17
8155
VDD
VCC
RDY
VSS
AD
PA
PB
PC
CLK
IOW
RD
WR
RST
ALE
IO/M
A8
A9
A10
CE
PROG
T/IN
T/OUT
CS0
CS1
CS3/
CS4
CS5
J3
J4
J5
PORT 0
PORT 1
PORT 8
PORT 9
PORT 21H
PORT 22H
PORT 23H
PORT 29H
PORT 2AH
PORT 2BH
TIMER OUT/
TIMER IN

Small System

The schematic in Figure 3 is of a complete microcomputer with only 6 ICs. The system contains its own serial I/O communication lines, multi-level interrupt, two programmable timers, and power-on reset. System capacity is 512 bytes of RAM, 4K bytes of PROM, and 76 lines of programmable I/O.

Block Move, Block Search

In a large system application high speed block moves may be necessary. The addition of an 8257 Direct Memory Access (DMA) controller and an 8255 Programmable Peripheral Interface (PPI) device with some miscellaneous logic removes the task from the 8085 and results in a very high speed capability. The addition of an eight bit comparator also permits block searches. (See Figure 4.)

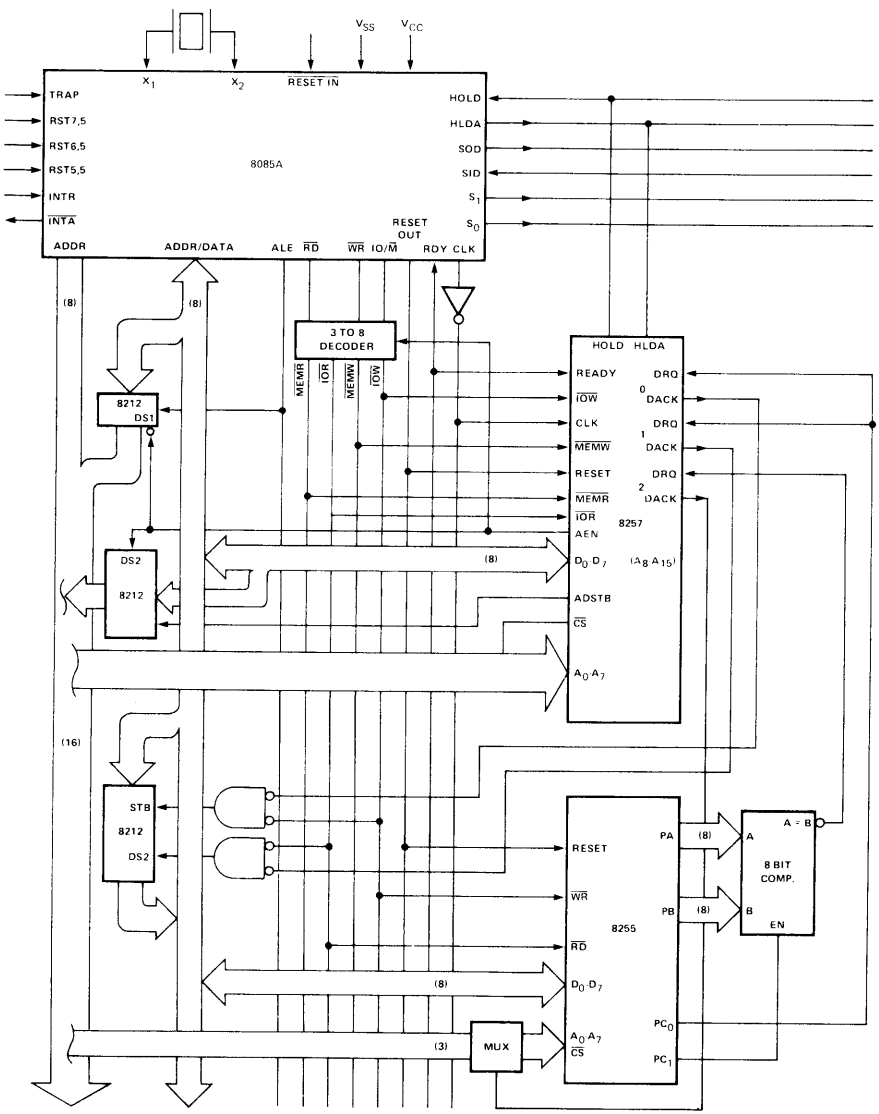


FIGURE 4. BLOCK MOVE, BLOCK SEARCH addition of an 8257 Direct Memory Access (DMA) controller and an 8255 Programmable Peripheral Interface (PPI) device permits block searches, high speed block moves. (2.5 μs/word).

Basic operation, for a block move, is that the CPU loads the 8257 with the starting address of the source block and the length* of the block into Channel 0. Channel 1 is programmed with the starting location of the destination block and the length. A bit in Port C of the 8255 is set by the CPU which initiates a DMA request on Channels 0 and 1. Because the 8257 is initialized to the rotating priority mode, the first DMA cycle is from Channel 0 which latches the data from the first location of the source block into the 8212. The second cycle will be from Channel 1 which will store the latched data into the first location of the destination block. The next cycle will return to Channel 0 and the sequence will start over again until the length (terminal count) is reached. Programming the 8257 stop bit insures that each channel will be disabled when its respective terminal count is reached.

This configuration also supports a block fill. DMA Channel 0 points to a location containing the fill value and has a length of one. Channel 1 points to the starting location of the destination block and contains the length. When the sequence is initiated the value will be loaded into the latch by Channel 0. Channel 0 reaches TC and is disabled. Priority rotates to Channel 1 which will repeatedly write into the destination block the value stored in the latch until TC is reached.

Block search operations use the 8-bit comparator and Ports A & B of the 8255 and Channel 2 of the 8257. The CPU loads Port B with the search value and the DMA channel with the search area (starting address and length). A Port C bit initiates the DMA READ request. Channel 2 DMA Acknowledge sets Port A of the 8255 up as the receiver for the DMA READ cycle by multiplexing A_0, A_1, and CS. Each cycle of the DMA then loads Port A with the value of the pointed-to location in the block. When Port A equals Port B, the output of the comparator will gate off the DMA request. The requesting program can now read the Channel 2 address which is pointing to the search value plus one. However, if the status register of the 8257 indicates that TC of Channel 2 has been reached, then no match was found.

*(The value loaded into the low-order 14-bits of the terminal count register specifies the number of DMA cycles minus one before the Terminal Count (TC) output is activated. For instance, a terminal count of 0 would cause the TC output to be active in the first DMA cycle for that channel. In general, if Length = the number of desired DMA cycles, load the value Length-1 into the low-order 14-bits of the terminal count register.)

RST 7

On the 8080A/8228 system if one tied $\overline{INTA}$ out of the 8228 to + 12 volts through a 1KΩ resistor, the 8228 would generate a RST 7 instruction to the 8080A upon interrupt. This was a very inexpensive mechanism.

The 8085A has expanded this facility with the RST 5.5, 6.5, 7.5 inputs but is not compatible with the RST 7 generated by the 8228. (Figure 5) To maintain this compatibility it can be achieved by adding an 8212 which will force a RST 7 instruction into the bus upon interrupt acknowledge ($\overline{INTA}$). (Figure 6)

RESTART	VECTOR LOCATION
RST 7	38_{16}
RST 5.5	$2C_{16}$
RST 6.5	34_{16}
RST 7.5	$3C_{16}$
TRAP	24_{16}

FIGURE 5. ADDITIONAL 8085A INTERRUPTS

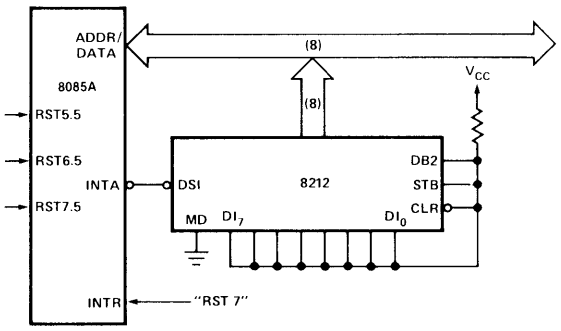


FIGURE 6. 8085A "RST 7" IMPLEMENTATION

SECTION 2
DETAILED APPLICATION EXAMPLES

Memory Addressing

One of the necessary functions of the microprocessor bus is to interface with the memory where the program is stored. ROM and EPROM memories are typically used to store programs while static and dynamic RAMS are generally used for data memory. The following discussions cover the interfacing to be used for these types of memory.

ROM - EPROM ADDRESSING

Later in this Appendix a section is devoted to an approach for developing a chart showing memory device compatibility for the 8085A. However, there is one area not included that will be discussed here, that is, unbuffered interfacing to standard ROM or EPROM memories. To use an unbuffered interface to ROM or EPROM it is necessary to understand a particular characteristic of the 8085A.

The 8085A has a period of time, T4 through T6 of the op code fetch cycle and certain instructions, where addresses A8 through A15 are undefined. Be careful about this. Not having addresses stable and using an address select method that would randomly turn on memory devices will cause bus contention and reliability problems in the unbuffered system. In the memory compatibility section of this Application Note, a minimum (unbuffered MCS-85 family and medium system (at least one level of buffering) configurations are considered. These configurations do not have bus contention problems. In the minimum system only MCS-85 components will be discussed where addresses are latched on the falling edge of ALE, thus ignoring any extraneous address transitions. The medium system is assumed to have data buffers that are enabled only at the proper time, thus again preventing any bus contention problems. What about the user who wants to use standard ROM or EPROM without buffering?

As an example let's look at Intel's ROM/EPROM family (Fig. 7) and develop a system block diagram. This system should allow upward compatibility for these particular devices and avoid any bus contentions due to undefined addresses. In Figure 8 a traditional decoding scheme is shown that uses the time difference between tacc (address access) and tco (chip select access) to allow for decoding of the EPROM/ROM to be selected. Connecting only these signals, however, in an unbuffered system will result in data contention because of the spurious addresses during opcode fetch. The proper interconnect for this type of interface is shown in Figure 9 where an output enable ($\overline{OE}$) signal will prevent any bus contention. This output enable is controlled by the read control signal, $\overline{RD}$, of the 8085A. This signal only occurs after addresses have stabilized.*

Note also that a PROM is recommended for the decoding function vs. an 8205 (1 of 8 decoder). Why? This PROM allows the user to easily upgrade his system to the 32 and 64K versions with minimum rewiring. As seen in Figure 3, only 4 pins are being altered (18-21) in the Intel ROM/EPROM family to allow for this upward compatibility. All a user would need to do is initially design his layout for 28 pin devices, thereby allowing total flexibility from 8K through 64K with the ease of only changing a decoding PROM and a few wires.† Application Note AP-30 can be ordered at no charge which fully discusses the application of Intel's 5 Volt EPROM and ROM family for microprocessor systems.

*Both $\overline{RD}$ and $\overline{WR}$ signals should be pulled up to +5V through a resistor to avoid random selection during 3-state.

†Another method is shown later in Figure 15 that facilitates the use of a decoder, such as the Intel 8205.

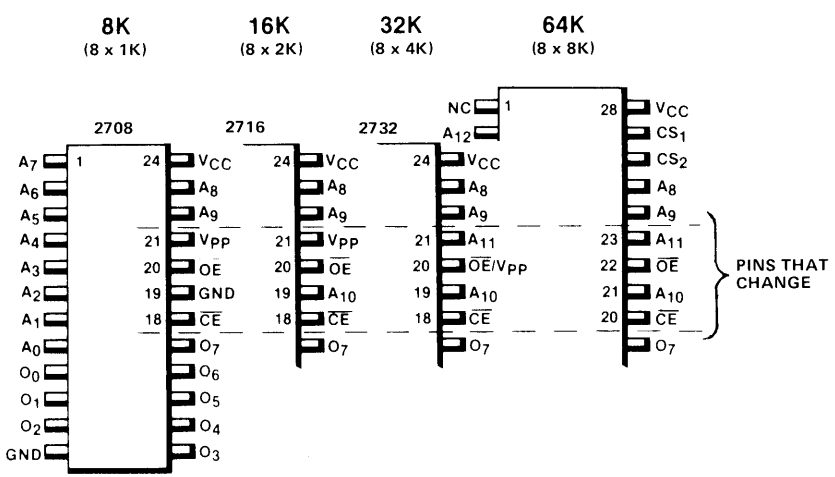


Figure 7. Intel® EPROM/ROM Compatible Family

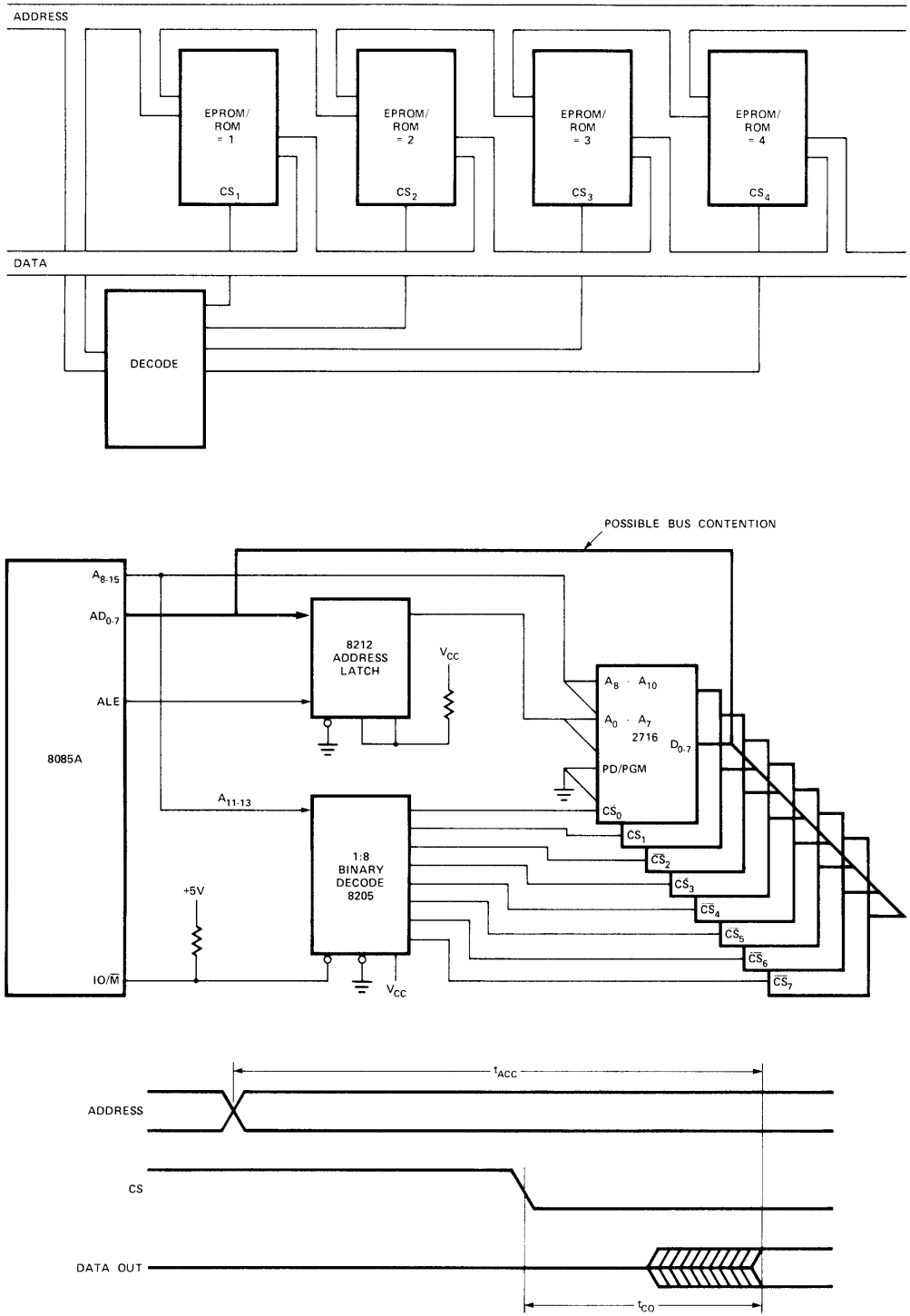


Figure 8. Traditional 16K EPROM System

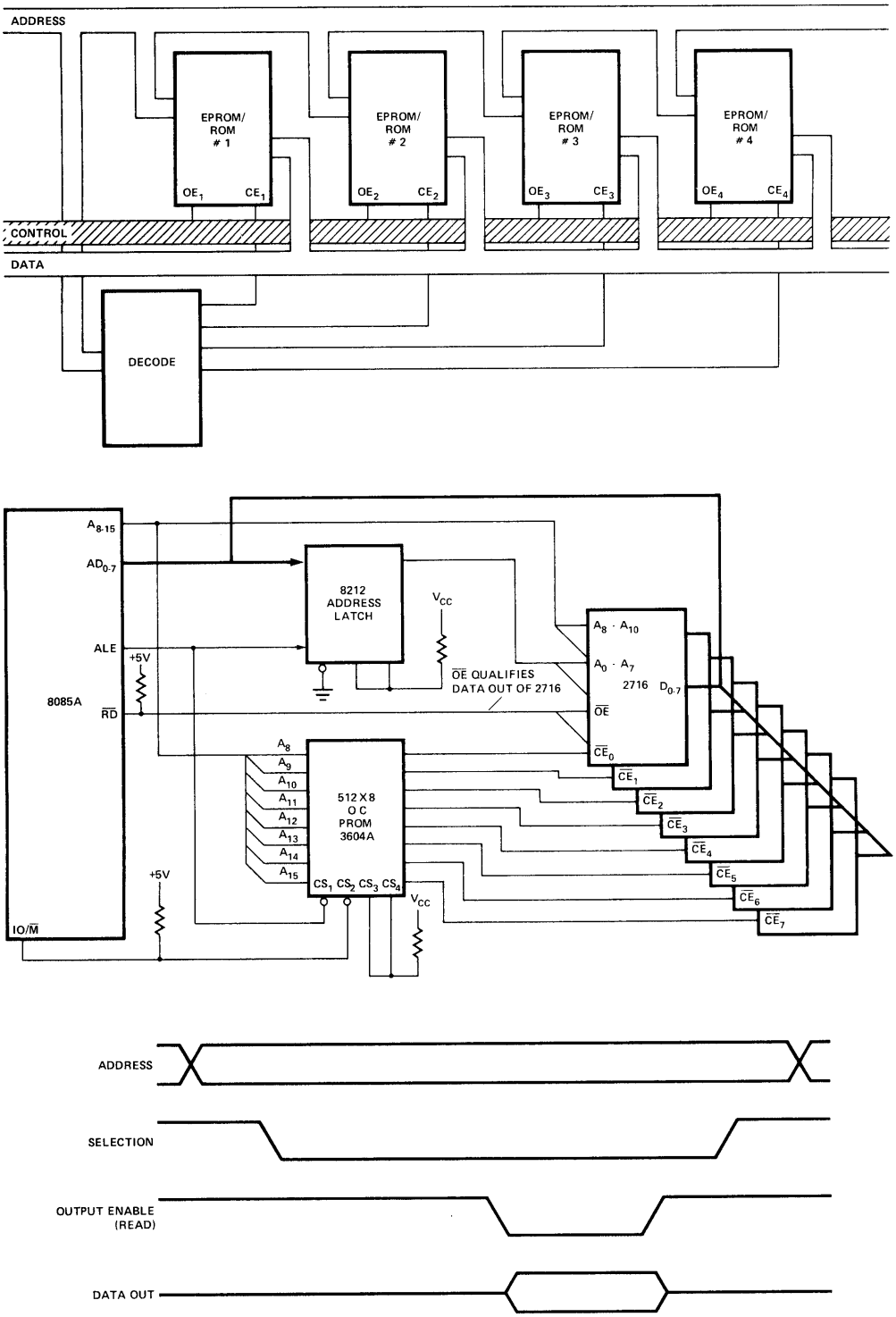


Figure 9. Correct 16K EPROM System

STATIC MEMORIES

The same consideration must be applied to standard static memories as with the ROMs/EPROMs in an unbuffered system. Memory device selection must be qualified by a memory read or write to prevent spurious selection. Some Intel static RAM devices have an Output Enable for this purpose, such as the 2142 (1k x 4). This part was designed to be specifically used with a microprocessor bus. For other standard static RAMs, the chip selects must be qualified by $\overline{RD}$, $\overline{WR}$ or ALE to prevent random selection.

DYNAMIC RAM INTERFACE

An earlier Intel Application Report (APR-1) extensively covered dynamic RAM interface with different types of memory and refresh in the MCS-80 system. This dynamic RAM section was taken from the most memory intensive example in APR-1, the 2116, modified to be compatible with the 8085A bus. These minor modifications are such that an 8080 system can be converted without much trouble. Before discussion of this section, however, a strong word of advice is in order. At about the same time this Application Note is published, Intel will be sampling an 8202 dynamic RAM refresh controller which does all dynamic RAM interfacing (except the data bus) and refreshing in *one* packaged component. It is highly recommended that the reader investigate this before using the attached schematic. Reading this section will still be useful in terms of understanding the 8085A bus.

This section uses the APR-1 2116 (multiplexed address 16K) example modified for the 2117-4 dynamic RAM. These devices have some differences from the 2116. One is that the output is not latched and is 3-stated during a write operation. This allows a user to tie both the data in and data out pins together at the device and at the data buffers, saving board traces. The 2117 also have hidden refresh capabilities where if $\overline{CAS}$ is held low, $\overline{RAS}$ can be toggled to refresh the device.

The schematic shown in Figure 10 is aimed at a high performance, relatively inexpensive solution (disregarding the 8202). Refresh circuitry is not shown, but can be implemented in a variety of ways. This will be discussed later in an upcoming section. In this refresh section, code for a simple, very low cost refresh controller that requires no special hardware, other than an 8155 timer, is presented.

For system timing, a 4x clock is used to obtain the resolution necessary to provide the clocks for the multiplexed address 2117's. Other solutions are possible with delay lines, one shots, etc., but are relatively expensive and don't provide for a nice baud rate source for any peripherals that may be in the system as does this 4x clock. Another approach can use the clock edges from the 8085A CLKOUT to interface to dynamic RAM. To facilitate this type of approach, Clock related timing parameters are listed later in this note.

To aid in understanding the operation of this circuit, the explanation is broken into a discussion of the main signal paths. 2117-4 Spec compatibility with the 8085A will be discussed in detail in the dynamic RAM section of the Memory Compatibility section.

Addresses

The lower 14 addresses (A0-A13) are used to select one of the 16,384 8-bit bytes in each 16K byte data bank. The lower 8 of these 14 addresses (A0-A7) flow through an 8212 and are latched by ALE, effectively demultiplexing the address/data bus. These lower 8 addresses with the next 6 (A8-A13) enter the 3242 multiplexer/refresh controller. The Row Enable of the 3242 controls which half of the addresses are presented to the dynamic RAM memory. Looking at the row enable on the 3242, it is seen that the row and column addresses are swapped with respect to convention. The higher order addresses are used as row addresses and the lower order addresses are used as column addresses. This does not create problems because this is invisible to the CPU. Refreshing is done properly as the 3242 controls the addressing for this. The upper two address lines (A_{14}-A_{15}) are decoded to qualify one of the four $\overline{RAS}$ (Row Address Strobe) lines to select one of the four 16K byte data banks of memory.

Cycle Requests

Cycle requests are generated from several sources; ALE automatically initiates a request when S1 indicates that there is a read taking place (flip-flop C), $\overline{WR}$ during write cycles (D) and refresh delayed (Q output of refresh flipflop (B)) when there is a refresh. ALE is used to start a read (qualified by S1) to provide ample time for access from the memories. This cycle request signal (A) immediately creates a $\overline{RAS}$ and starts a timing chain (74S174 shift register (E)) to generate the remaining signals. Synchronization between this cycle request pulse and the 4x clock is accomplished by the first D flip-flop in the 'S174 shift register (timing chain).

$\overline{RAS}$/$\overline{CAS}$

When $\overline{RAS}$ is enabled by a cycle request, it is qualified with either a refresh request (all $\overline{RAS}$'s turn on) or the decoded upper two bits of the address bus. A careful reader may question whether address is valid prior to $\overline{RAS}$ being enabled. This question can be answered by noting that the 8212 passes the address through before the falling edge of ALE latches it. T_{AL}† (115 ns for 320 ns 8085A processor cycle), which is the time from address to the falling edge of ALE, gives ample time for addresses to be valid at the 3242 outputs before $\overline{RAS}$ is valid. $\overline{RAS}$ is extended past the clearing of the cycle request flip-flop by ORing this enabling signal with a tap from the D flip-flop shift register.

$\overline{CAS}$ (Column Address Strobe) is produced between 123 and 164 ns after $\overline{RAS}$, depending upon when the first D flip-flop in the shift register synchronizes with the cycle request signal (C). Since this is greater than the specified maximum delay from $\overline{RAS}$ to $\overline{CAS}$, this memory system is $\overline{CAS}$ access limited and $\overline{RAS}$ access no longer has any meaning. The $\overline{CAS}$ tap can't move up one D flip-flop to provide more time for memory access as this would not provide sufficient data set up time with respect to $\overline{CAS}$ during a write.

†Note that T_{AL} now only applies to the high order address byte. TALL, for the lower address byte equals 90 ns. This was done to allow for additional T_{RAE} time for data float.

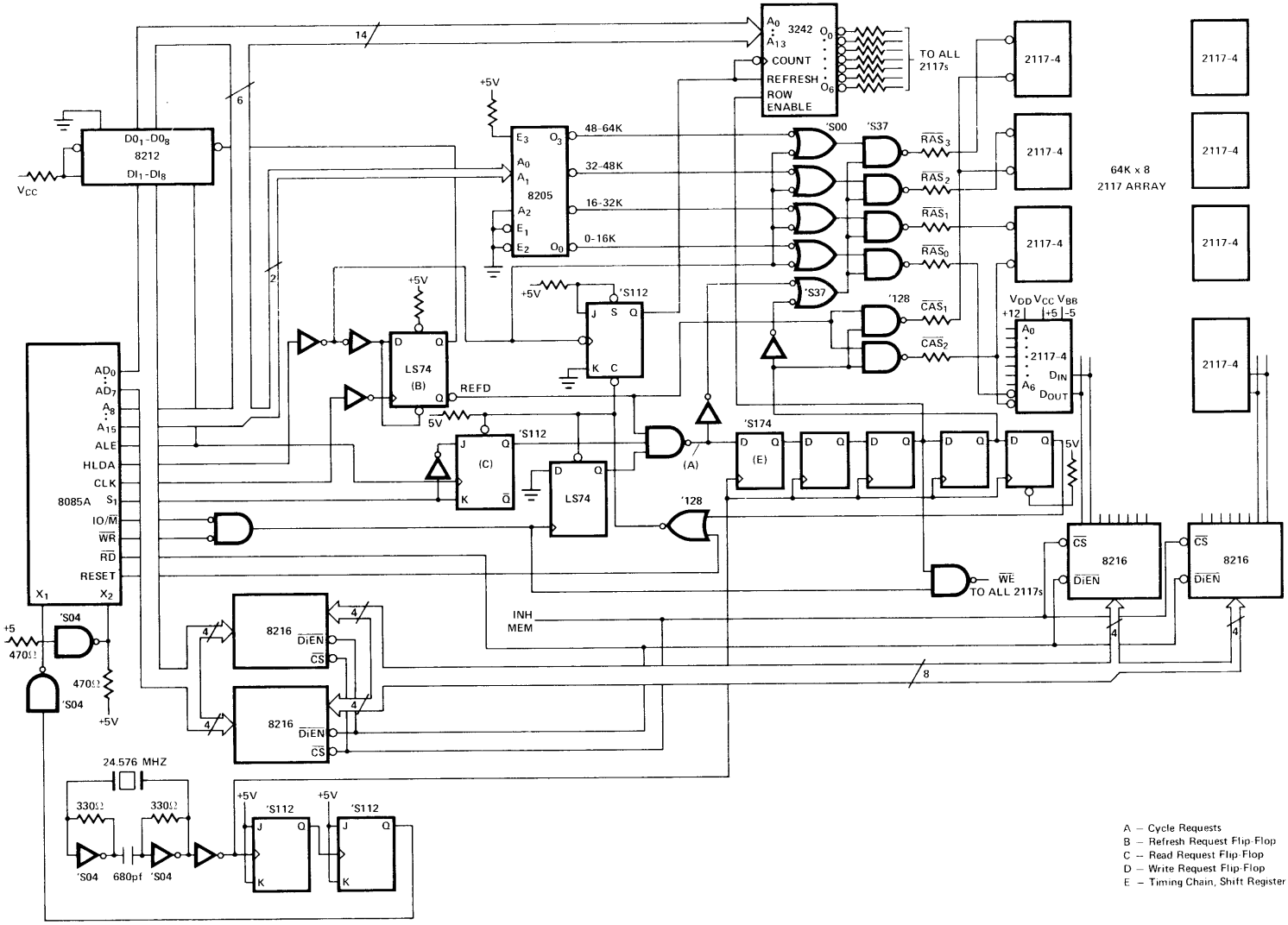


Figure 10. 8085A-2117 Dynamic RAM Interface

Data

The data path to the 2117s is through two sets of buffers to account for memory being off board. To determine bus timing it is helpful to know that Write data is not guaranteed to be valid from the 8085A until 40 ns after the leading edge of the write control signal. On account of this and the delay times for the buffers it is necessary to delay the cycle request on a write until the $\overline{WR}$ signal goes low. The solution shown still does not require wait states. An inhibit memory signal is also involved. This is useful when using memory address space overlap such as the case with bootstrap ROM (which would be necessary in this system if a full 64K of dynamic RAM is used).

Refresh

Dynamic RAMs are generally refreshed in two different modes; burst (i.e., all at once every 2 ms) and distributed (one row every (2 ms/number of rows) period of time). The schematic shown provides for a distributed refresh where refresh requests are applied to the Hold request input of the 8085A (not shown). This signal needs to occur at least once every 15 µsec ((2ms/128 rows to be refreshed) - HOLD to HLDA delay) and can be generated through a baud rate timing chain, Intel 3222, one shots or other similar devices. Another approach to refresh could qualify the refresh cycles with program fetch cycles (use status lines). If program memory is in static RAM or ROM and the dynamic RAM bus can be isolated, refresh cycles can be performed with no overhead. Instead of using the HOLD feature of the 8085A, refresh can be hidden in the program fetch and decode. Further considerations for refresh include proper handling of resets and excessive hold times from other peripherals to be certain the memory is being refreshed adequately.

Some applications don't require high CPU efficiency and require a very inexpensive method to refresh their dynamic RAM. Since writing, reading or performing special refresh cycles all refresh a particular row, why not do "dummy" reads to refresh? To use this technique memory must be mapped on a one to one correspondence with the address space. This will allow the programmer to read one byte in each physical row in the 2117s, thereby refreshing that row. A simple software routine can be devised to refresh 16K bytes of RAM. If more dynamic RAM than this is desired it can be accomplished by specially enabling all the desired RAS signals via an 8085A output port. First let's analyze how many CPU cycles are available in the 2ms period:

2ms/(320 ns/cycle) = 6,250 cycles
for 8085A@ 3.125 MHz

2ms/(200 ns/cycle) = 10,000 cycles
for 8085A-2@ 5.0 MHz

If there is a convenient component that can count 8085A cycles (8085A CLKOUT) and interrupt the 8085A, you're home free. An example of such a device is the 8155 in the MCS-85 family. On the 8155 one can use the $\overline{TO}$ (timer out) pin to interrupt the CPU everytime a refresh needs to be performed and an interrupt service routine could dummy read 128 consecutive locations and return to CPU operation. (128 reads are necessary to completely refresh the full 16K bytes of 2117 memory.) The highest priority interrupt should be used for this to insure that refresh occurs. Figure 11 is an example program to perform this burst dummy read refresh. This routine basically uses 64 pops of the stack, each reading two consecutive locations in the memory. Note that this routine destroys the contents of registers B, C and D in the 8085A. The user may want to save these registers in the routine before performing the software refresh. If memory space is more valuable than CPU efficiency, the POPs can be performed in a loop instead of a string, saving additional memory.

This routine requires 690 cycles which is about 11% of the available 8085A CPU cycles, or 7% of the available 8085A-2 cycles. If this is acceptable and there is a counter available, you can't find a cheaper way to do refresh. Note that as processor speeds become faster, this overhead becomes proportionately less and more attractive as an alternative. Again, as with any refresh routine, reset and excessive holds must be dealt with to guarantee proper refresh.

DMA (Direct Memory Access)

DMA is becoming more common in the microcomputer system for many applications. Some examples include the 8271 floppy disk controller and refreshing a CRT via an 8275 CRT Controller. It is always helpful to reduce the overhead of the DMA (as DMA can tie up the system bus) whenever possible. In many applications, where program memory is resident in ROM or PROM, DMA cycles can be performed in coincidence with op code fetch. This will make them invisible to the CPU as described for Refresh in the Refresh section of the 2117 dynamic RAM example.

In the dynamic Ram system, Refresh requests can be made on the DMA controller via the DRQ lines, with the 8237 in a rotating priority mode to insure refreshing is done. Another technique would be to devise an arbiter for DMA and refresh requests at the processor hold input. With this technique the designer must not allow DMA to monopolize the bus when refresh is needed.

```
During initialization:
   MVI          A, D5H   SET TIMER COUNT TO 5550* FOR REFRESH COUNT
   OUT     TIMER MSBYTE
   MVI          A, A4H   INTERRUPT CPU AT TO (TIMER OUT)
   OUT     TIMER LSBYTE
   MVI          A, C0H   START COUNTER, PLACE C0 IN 8155 STATUS REG.
   OUT  TIMER COMMAND
   Program

AT RST 7.5 RETURN ADDRESS      CALL RFRS      (REFRESH SERVICE)
TOTAL      #CYCLES
CYCLES
            10          RFRS:  LXI HL, 0      SAVE STACK POINTER IN HL
            10                 DAD SP
 30         10                 LXI SP, 0080   32K - 48K REFRESH

            10                 POP BC
            10                 POP BC         REFRESH, DUMMY READ
                               .
                               .
                               .              64 TIMES
                               .
640
             6                 SPHL           RESTORE STACK POINTER
             4                 EI             ENABLE INTERRUPTS
 20          10                RET            RETURN

690          TOTAL CYCLES      (round up to 700)
```

*6,250 available cycles - 700 to do refresh. Counter should count 5550 = 15A4H for 8085A; for 8085A-2 must count 10,000-700 = 9300 = 2454H. To set counter to automatic reload, most significant bits in timer of 8155 must be set to 1. Therefore, for 8085A use D5A4H and for 8085A-2 use E454H.

Figure 11. Software Refresh

The standard technique for interfacing the 8085A processor to the 8237 DMA controller is shown in the MCS-85 User's Manual and is reproduced in Figure 12. This configuration is set up to interface with standard memories or peripherals, i.e., ones that don't share their data bus with addresses, not the MCS-85 family components (8155, 8355, 8755A, etc.). DMA is unlikely with these MCS-85 components as they are intended for minimum system applications. If the system has both MCS-85 and standard addressed components, and DMA is used for the standard addressed components, ALE must be or'ed with ADSTB from the 8257. This is necessary to deselect the MCS-85 components from the bus. Due to the latching feature of the MCS-85 components, bus contention may result if this is not done and DMA tries to use the bus.

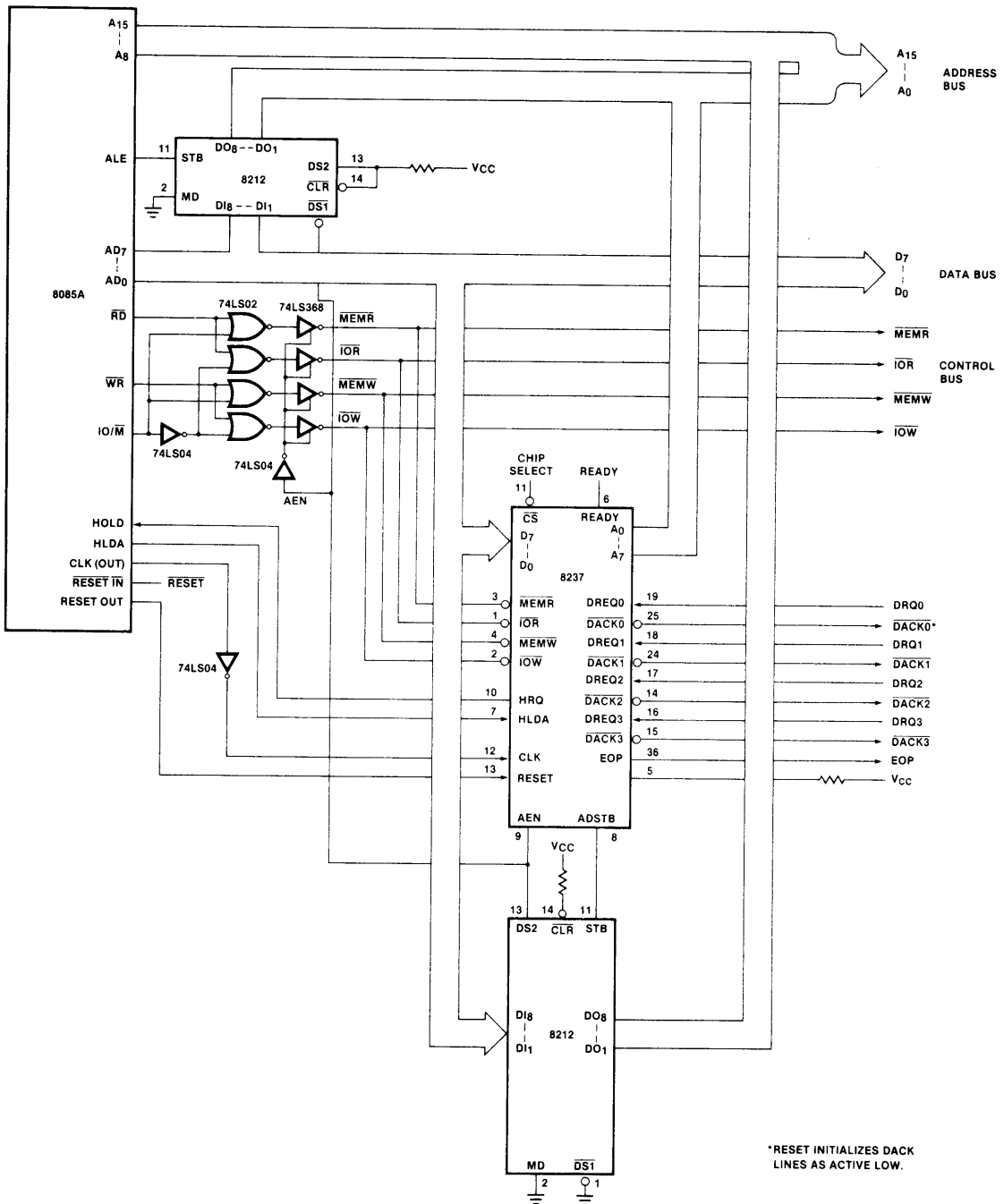


Figure 12. Detailed System Interface Schematic

SYSTEM TIMINGS

8085A CLK-IN vs. CLK-OUT vs. Control Timings

This section shows timing characteristics that relate the input clock to the control signals and the output clock. These timings can be treated as constants, that is, within the normal operative range of the processor, they are cycle or 8085A, A-2 speed independent.

Be careful about manipulating the timings given in this section with the specifications in the data sheet. The specifications on the 8085A, A-2 are *not* mutually exclusive; that is, you can't add minimums to minimums and obtain a valid minimum for some other timing parameter. Where the timing parameter is specified directly, this takes precedence over any other method you come up with to find that specification (through adding and subtracting others). This was not done to confuse the user, but to provide him with the most optimal timings for his system!

To understand the timing parameters in this section it would be helpful to understand how the internal signals are generated in the 8085A. Referring to Figure 13, it is seen that the rising edge of the X1 input causes flip-flop A to toggle. From this flip-flop two internal signals are generated that drive all functions in the 8085A, A-2 and produce the output control signals and clock. Referring to Figure 15, it is seen that clock output is derived from the internal $\phi 1$ signal in the schematic of Figure 14. This output signal is a MOS output unlike the bipolar outputs of the 8224 in the 8080A system. This restricts the user to the loading limitations of a MOS driver (for further details see bus loading section). The rest of the output control signals with their respective internal controlling edges are also shown in Figure 14.

Since the path between the X1 input and the clock output can have a considerable amount of variance, the relationships of these two clocks vary significantly. Figures 15 and 16 are a set of timing diagrams illustrating the relationship of the clock input to the clock output to the various control signals. For designs that require these relationships to synchronize different systems, components, etc; the designer must allow for these variances in the relationships.

Parameter	Description	Min	Max	Units
$t_{\phi AL}$	Time from C.F. to next Address valid (A_0 - A_7)		130	ns
$t_{\phi ALU}$	Time from C.F. to next Address valid (A_8 - A_{15} only)		70	ns
$t_{\phi AT}$	Time from C.F. to present Address remaining valid (A_8 - A_{15} only)	−20		ns
$t_{\phi CL}$	Time from C.F. to control low (L.E.)	−10	60	ns
$t_{\phi CT}$	Time from C.R. to control low (T.E.)	−10	60	ns
$t_{\phi DL}$	Time from C.F. to Data Out becoming valid		65	ns
$t_{\phi DT}$	Time from C.F. to Data Out remaining valid	−20		ns
$t_{\phi DS}$	Data-in set up time to C.R.	100		ns
$t_{\phi DH}$	Data-in hold time to C.R.	0		ns
$t_{\phi LL}$	Time from C.F. to ALE high (L.E.)	−60	0	ns
$t_{\phi LT}$	Time from C.R. to ALE high (T.E.)	0	70	ns
t_{XKF}	Time from X_1 input to C.F.	30	150	ns
t_{XKR}	Time from X_1 input to C.R.	30	120	ns

C.F. = Clock Falling, C.R. = Clock Rising, L.E. = Leading Edge, T.E. = Trailing Edge

NOTE: These numbers are guaranteed by design and are not tested by Intel.

8085A, 8085A-2 Clock Parameters

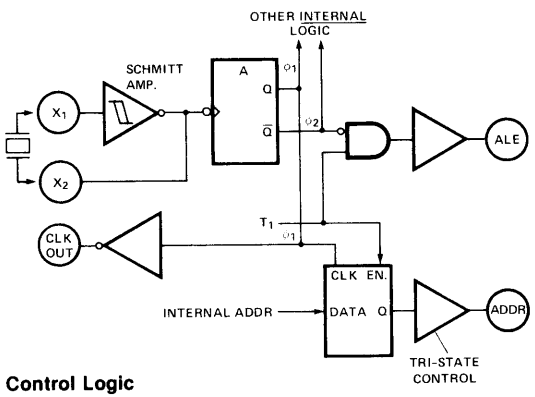


Figure 13. Clock and Sample Control Logic

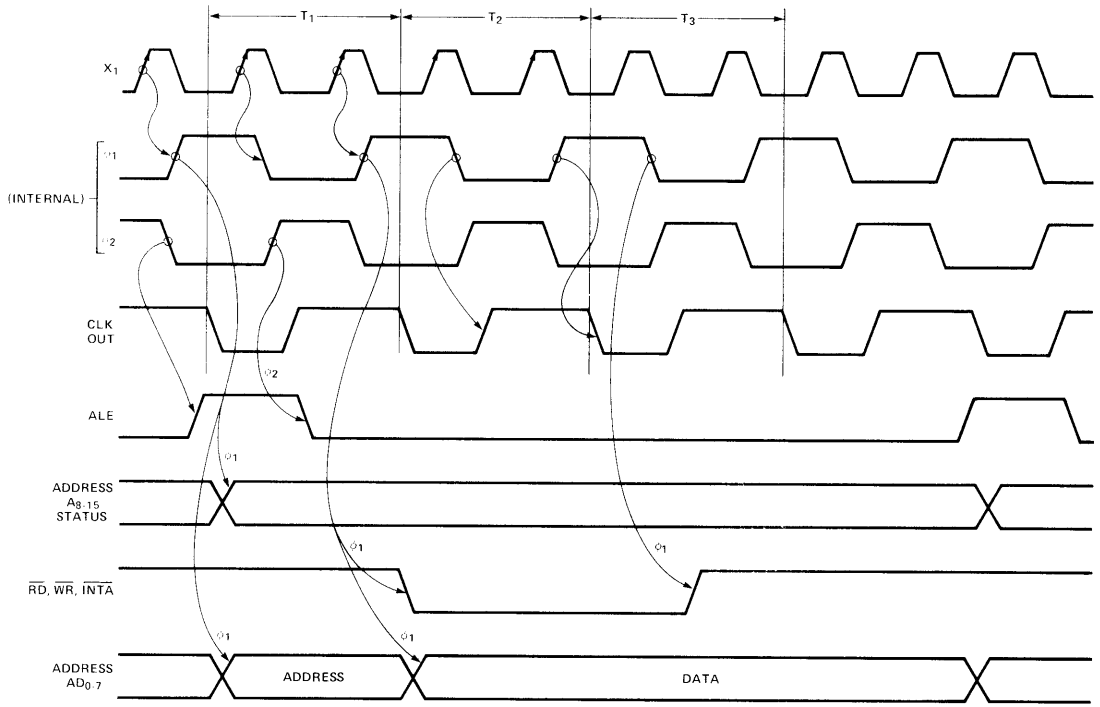


Figure 14. Clock In (X1) to Output Relationship

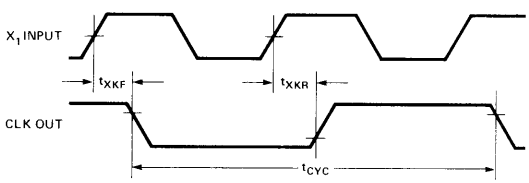


Figure 15. 8085A-2 Clock In/Clock Out Timing

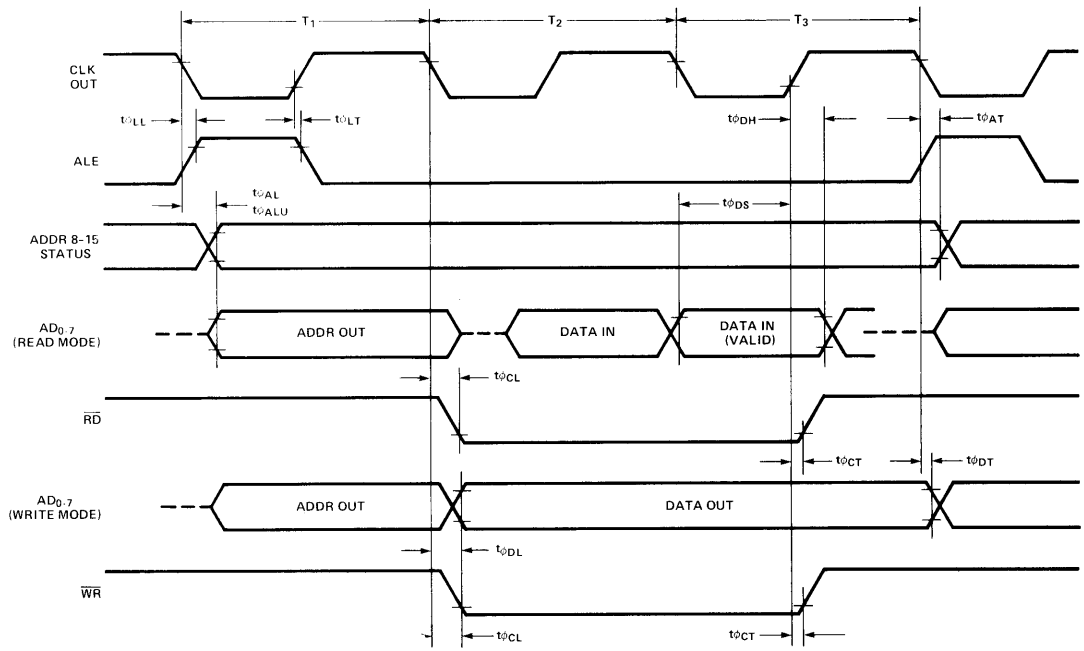


Figure 16. 8085A-2 Clock Related Timing

3.125 vs. 5 MHz Considerations

The 8085A (with maximum internal clock frequency of 3.125 MHz) and 8085A-2 (5 MHz) have some differences in their bus operation. There are two sets of peripherals that can be used with both the 8085A and A-2. There are the dedicated peripherals in the MCS-85 family that directly interface with the 8085A, A-2 bus and the standard MCS-80 peripherals that Intel also provides. The standard peripherals that are denoted 825X-5 (also the 8251A and 827X peripherals) are peripherals that can be used with an 8085A or 8085A-2. In the 8085A-2 system a wait state is required for proper I/O operation, but even with this wait state system speed is still 30% higher than the 8085A without wait states. An example wait state generator for this purpose is shown at the end of the peripheral compatibility section in this Application Note (Figure 19).

The main timing differences to consider when using an 8085A vs. an A-2 are listed in Table 1.

Cycle dependent timings are listed in Table 2. These are very useful when the user is not operating at the full bus speed. Remember that each 8085A, A-2 device divides its clock input frequency by 2. Therefore, a 10 MHz crystal will produce a 200ns output cycle (denoted as T in the cycle dependent timings). A timing diagram showing the relationships of the timing parameters given in Table 2 can be found on the data sheets.

—Clock (crystal) requirements. The 8085A, A-2 requires the following crystal specifications to run at top bus speed:
8085A 6.25 MHz frequency, parallel resonant, fundamental, 10 mwatt drive level, RS < 75 ohms, CL = 20-35 pf, and CS < 7 pf.
8085A-2 10 MHz frequency, all other specifications the same as 8085A.
—Memory and Peripheral Compatibility - Discussed in detail in upcoming sections.
—Cycle dependent timings (Table 2)

Table 1. 8085A vs. 8085A-2.

Parameter	3.125 MHz (8085A) (ns) Min	Max	Cycle Dependencies	5 MHz (8085A-2) (ns) Min	Max	Cycle Dependencies
tcyc	320	2000		200	2000	
t1	80		1/2T—80	40		1/2T−70
t2	120		1/2T—40	70		1/2T−50
tr		30			30	
tf		30			30	
tAL	115		1/2T−45	50		1/2T−50
tLA	100		1/2T−60	50		1/2T−50
tLL	140		1/2T−20	80		1/2T−20
tLCK	100		1/2T−60	50		1/2T−50
tLC	130		1/2T−30	60		1/2T−40
tAFR		0			0	
tAD		575	(5/2+N)T−225		350	(5/2+N)T−150
tRD		300	(3/2+N)T−180		150	(3/2+N)T−150
tRDH	0			0		
tRAE	150		1/2T−10	90		1/2T−10
tCA	120		1/2T−40	60		1/2T−40
tDW	420		(3/2+N)T−60	230		(3/2+N)T−70
tWD	100		1/2T−60	60		1/2T−40
tCC	400		(3/2+N)T−80	230		(3/2+N)T−70
tCL	50		1/2T−110	25		1/2T−75
tARY		220	3/2T−260		100	3/2T−200
tRYS	110			100		
tRYH	0			0		
tHACK	110		1/2T−50	40		
tHABE		210	1/2T+50		150	1/2T+50
tRV	400		3/2T−80	220		3/2T−80
tAC	270		T−50	115		T−85
tHDS	170			120		
tHDH	0			0		
tINS	360		1/2T+200	150		1/2T+50
tINH	0			0		
tLDR		460	2T−180		270	4/2T−130

Where T = tcyc and N = the number of wait states that are incorporated.
All mathematical operations in Table 2 are performed from left to right, except where qualified with parenthesis.

Table 2. 8085A and 8085A-2 Cycle Dependencies

Memory Device Compatibility

Determining What Memory to Select For Your Application

When developing a system which will use sufficient memory to require buffering (see the capacitive loading section to determine when it is needed), it is important to understand how to select the slowest, lowest cost memory and still be compatible with the bus timings with minimum wait states. A generalized procedure has been developed in the following section for determining the memory access needed for different applications and the number of wait states required (if any). In general the amount of time available for accessing the memory can be obtained from the following formula:

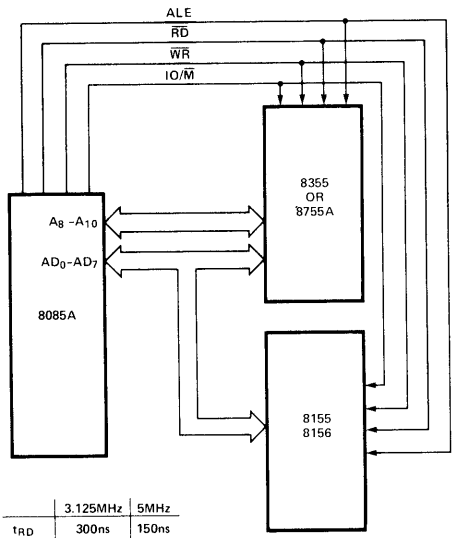
Available memory access = 8085A access time (from control signal of interest) - Buffering/Decoding delay (to and from memory)

The three main "control" signals of interest which determine memory access are that of t_{RD} (read to valid data in), t_{AD} (valid address to valid data in) and t_{LDR} (address latch enable to valid data in). When dealing with different types of memories, one or more of these signals becomes important.

Even though memory access compatibility is probably one of the most important parameters to consider, as this is directly reflected in the price of the memory, it is not the only parameter that is important. Some of the other major timing considerations are as follows:

WRITE ENABLE - Is the write enable signal sufficiently long to guarantee a write?
Is data set up properly with respect to this write to be compatible with the memory's requirements?
Is data held long enough?

DATA FLOAT - Does your system have sufficient margin to prevent bus contention?
(i.e., Does the memory let go of the data bus in time for the processor to use it? Remember that the 8085A shares its Data Bus with the lower 8 addresses.)



	3.125MHz	5MHz
t_{RD}	300ns	150ns
t_{AD}	575	350

Figure 17. Minimum System

We will first go through the minimum system which can be represented by the dedicated set of components Intel has developed for the 8085A (Fig. 17). The two timing specs were taken from the data catalog for t_{RD} and t_{AD} (t_{LDR} is irrelevant here). Looking at the 8155/6 and 8355/8755A, a comparison can be made for the access times:

	8085A (3.125 MHz)	8155/6	8355/8755A
t_{RD}	300 (max)	170 (max)	170 (max)
t_{AD}	575 (max)	400 (max)	400/450 (max)

This shows that there is plenty of bus margin for the 3.125 MHz minimum application of the 8085A. Access time for the processor can be interpreted as the time from when the control signal is presented on the bus to the time when the processor will expect the data to be valid so it can sample it. Conversely, memory access times show the amount of time that will elapse between when it is told to present its information to when it actually does it. As long as the memory access spec is less than the processor access spec (minus appropriate buffering delays) the memory is access time compatible.

In more complicated systems where one level of data, address and control buffering is required (such as the case when there are many signal paths and device loading on one card), the delays of the latches and bidirectional drivers must be taken into consideration.

First consider a ROM, EPROM or static RAM configuration as shown in Figure 18. Using the generalized available memory access formula, t_{AD}, t_{RD} and t_{LDR} for the memory can be determined using the data sheet timing delays for the buffers.

	3.125MHz	5MHz
T_{RD}	250	100
T_{AD}	500	275
T_{LDR}	390	200

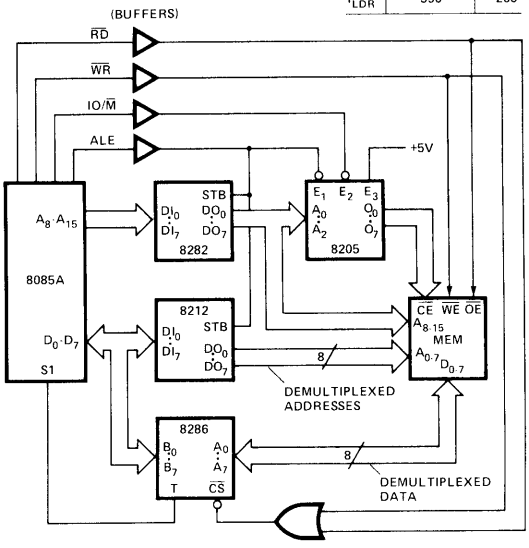


Figure 18. Medium Buffered System

t_{AD} MEMORY = t_{AD}8085A − (8282 + 8205 delay) − (8286 delay) + transitional gain due to buffering*

= t_{AD}85 − (T_{IVOV} + t--) − (T_{IVOV}) + t_{CAPB}*

= (5/2+N)T − 225 − 55 − 35 + 15

= (5/2+N)T − 300 (for 8085A)

(5/2+N)T − 225 (for 8085A-2)

where N = number of wait states and T = cycle time,

For minimum 8085A timing 500ns = t_{AD} memory

8085A-2 timing 275ns = t_{AD} memory

The 8085A timing parameter t_{AL} was not taken into consideration as the 8282 transfers information directly through without concern of the address latch enable. t_{RD} can be obtained in a similar manner.

The read signal $\overline{RD}$ goes through a buffer before it reaches the memory. This must be taken into consideration when calculating effective t_{RD} for the memory.

t_{RD} MEMORY = t_{RD} 8085A − (buffer delay) − (8286 delay) + transitional gain due to buffering

= t_{RD} 85 − (delay) − (T_{IVOV}) + t_{CAPB}

= (3/2+N)T − 180 − 30 − 35 + 15

= (3/2+N)T − 230 (for 8085A)

(3/2 + N)T − 200 ns (for 8085A-2)

*t_{CAPB} is additional time thrown back in for improvement in signal transitions. This is because buffering the signals reduces the capacitive loading considerably. The data sheet gives timings for maximum capacitive loading. Characterization has shown change in delay versus capacitive loading as .12 ns/pf min (under 20 pF loading) and .24 ns/pF max (under 150 pF loading). To take into consideration the effects of this loading two parameters are defined:

t_{CAPA} - delay for a signal to leave the old logic level

t_{CAPB} - delay for a signal to complete the transition from the old to new logic level

where t_{CAPA} = 1/2 t_{CAPB}

	MIN	MAX
t_{CAPA}	7 ns	15 ns
t_{CAPB}	15 ns	30 ns

In the memory compatibility calculations tCAPB min is added on as spec sheet values assume 150 pF loading and this system is not worst case, i.e., it has buffering that reduces this loading to approximately 20 pf. Since the CAP = 130 pF and change in delay versus capacitance is 1/2 ns/pF min, t_{CAPB}MIN = (.1 ns/pF) 130 pF = approx. 15 ns.

For minimum 8085A timing 250ns = t_{RD} memory

8085A-2 timing 100ns = t_{RD} memory

Therefore for t_{LDR}:

t_{LDR} MEMORY = t_{LDR} 8085 − (buffer delay) − (8205) − (8286) + t_{CAPB}

= t_{LDR} − (delay) − (t--) − (T_{IVOV}) + t_{CAPB}

= 2T − 180 − 30 − 20 − 35 + 15

= 2T − 250 for 8085A

= 2T − 200 for 8085A-2

For minimum 8085 timing = 390ns

8085A-2 timing = 200ns

To obtain memory access parameters for a multicard system (which would have buffering at both ends of the system bus), it is a simple matter of subtracting off the additional buffering delays.

With these timings a memory compatibility table can be developed from the data sheets (Table 3). With most of these memories it is relatively straightforward to determine the controlling signal used to select and enable the device. To illustrate this, listed below are the controlling signals of interest for the different memories as they are used in a typical configuration:

		Relevant Control Signal
RAM		
2114		
	Address access	- t_{AD} MEM
	Chip select access	- t_{LDR} MEM**
2142		
	Address access	- t_{AD} MEM
	Chip select access	- t_{LDR} MEM**
	Output enable	- t_{RD} MEM
ROM		

**Chip selects for these static RAMs need not be qualified with ALE. If 2114 or 2142 chip selects are generated directly from the address lines, the relevant timing is t_{AD} MEM.

	3.125 MHz	5 MHz
MINIMUM SYSTEM:		
STATIC RAM	8155/8156, (256x8) 8185 (1Kx8)	8155-2/8156-2 8185-2
ROM/EPROM	8355 (2Kx8) 8755A (2Kx8)	8355-2 8755A-2
BUFFERED SYSTEM:		
STATIC RAM	2114 (1Kx4) 2142 (1Kx4)	2114-2 2142-2
ROM/EPROM	2732 (4Kx8) 2716-2 (2Kx8)	2716-2**

*Contact Intel for high performance EPROM/ROM Family.
**With 1 wait state.

Table 3. 8085A, A-2 Memory Compatibility.

In general, t_{AD} MEM and t_{LDR} MEM are the parameters needed for chip enabling, selection and address access times, and probably are the most important considerations when determining which memory device to use. When there is an output enable, tRD MEM is also used. All relevant access times must be met by the resulting system configuration to be compatible.

This note will not attempt to generalize a procedure that deals with the interface to dynamic RAM, but the 2117 example shown earlier is described below. In the dynamic RAM system, many variables come into play upon which the memory access is dependent. Among these are refresh controllers, decoding, whether or not the system is designed for minimum hardware or maximum performance, and consideration for nonmultiplexed vs. multiplexed address dynamic RAMs.

For the Intel 2107C, which has nonmultiplexed addresses, tAD is the important parameter as it generates the chip selects and chip enables. However, with a multiplexed address part, things are different and both a $\overline{RAS}$ and $\overline{CAS}$ access time must be considered. Note that since $\overline{RAS}$ is applied before $\overline{CAS}$, $\overline{RAS}$ access time is effective only while the $\overline{CAS}$ signal stays within the specified $\overline{RAS}$ to $\overline{CAS}$ delay time. If it is not possible to do this, $\overline{CAS}$ access becomes the limiting factor for memory selection. Don't be mislead by the $\overline{RAS}$ to $\overline{CAS}$ maximum delay (tRCD: $\overline{RAS}$ to $\overline{CAS}$ delay time) spec'd on dynamic RAM data sheets! This maximum only applies to guarantee $\overline{RAS}$ access.

For a specific example the following shows how the speed versions were selected for previous 2117 dynamic RAM interface.

$\overline{RAS}$ path (from ALE)	approximate delay
5 gates	7 ns ea
1 Flip Flop	15 ns
(return path) 2 8216s	25 ns ea

$\overline{CAS}$ path (from ALE)	approximate delay
3 gates	7 ns ea
1 Flip Flop	15 ns
4 D Flip Flops	41 ns ea

t_{ACCESS} AVAILABLE FOR $\overline{RAS}$ =

$$t_{LDR} - 5(7) - 15 - 2(25) = 360 \text{ ns}$$

t_{ACCESS} AVAILABLE FOR $\overline{CAS}$ =

$$t_{LDR} - 3(7) - 15 - 4(41) - 2(25) = 210 \text{ ns}$$

Since $\overline{RAS}$ available time – $\overline{CAS}$ available time is greater than the spec value for $\overline{RAS}$ to $\overline{CAS}$ delay on all 2117 specs, $\overline{CAS}$ access becomes the limiting factor. A $\overline{CAS}$ access of 165ns of the 2117-4 is well within the time available.

To verify the other 2117 specs such that there is certainty that this sytem will play, a comparison can be made of the timing specs in the 2117 data sheet to the timings that result in the circuit configuration in Figure 12. When looking at the following timing comparisons, remember that the read cycle is initiated by the falling edge of ALE (Address Latch Enable) and the write from the falling edge of $\overline{WR}$ (Write). For descriptions of the parameters in Table 4, please refer to a 2117-4 data sheet. Delay assumptions used are shown in Table 5.

READ CYCLE	TAKEN FROM 2117-4 DATA SHEET MIN	MAX	DYNAMIC RAM CONFIGURATION MIN	MAX
tRAC		250 ns	Doesn't apply	
tCAC		165 ns	210 ns	
tREF		2 ms	Not Shown	
tRP	150 ns		279 ns	
tCPN	25 ns		472 ns	
tCRP	−20 ns		193 ns	
tRCD	35 ns	65 ns	Outside spec, CAS access limited	
tRSH	165 ns		177 ns	
tCSH	250 ns		300 ns	
tASR	0 ns		55 ns	
tRAH	35 ns		82 ns	
tASC	−10 ns		−4 ns	
tCAH	75 ns		205 ns	
tAR	160 ns		410 ns	
toff	70 ns		See Below*	
tRC	410 ns		720 ns	
tRAS	250 ns		307 ns	
tCAS	165 ns		198 ns	

*There are two parameters that the processor "sees". One is memory access, which has already been covered. The other is when the memory will let go of the bus. To show compatibility here, the following analysis is done:

2117 t_{OFF}	70 ns max
8085A t_{RAE}	150 ns min

Therefore compatible as $\overline{WR}$ is used to deselect the 8216's.

Table 4. Bus Compatibility Analysis (see Figure 11)

WRITE CYCLE	TAKEN FROM 2117-4 DATA SHEET MIN	MAX	DYNAMIC RAM CONFIGURATION MIN	MAX
tRC	410 ns		720 ns	
tRAS	250 ns		307 ns	
tCAS	165 ns		198 ns	
tWCS	−20 ns		34 ns	
tWCH	75 ns		164 ns	
tWCR	160 ns		287 ns	
tWP	75 ns		205 ns	
tRWL	100 ns		205 ns	
tCWL	100 ns		205 ns	
tDS	0 ns		23 ns **	
tDH	75 ns		Data held until next cycle	
tDHR	160 ns		Data held until next cycle	

**Data is not valid from the 8085A until 40 ns after $\overline{WR}$ falls.

Table 4. Bus Compatibility Analysis (see Figure 11) (Cont'd)

The numbers in Table 4 were obtained by using the following delay assumptions (Table 5) and very conservative techniques of obtaining minimum 8085A timings. Where no direct specification applied, minimum specs were added assuming 0 ns for any rise or fall times. This is more conservative than necessary. Another approach can be made from the clock related timings discussed in an earlier section.

	DELAY MIN	MAX	
Gates	0 ns	7 ns	
Flip Flops	0 ns	15 ns	
8216s	0 ns	30 ns	
D flip flop (Timing Chain)	41 ns	41 ns	
3242	0 ns	25 ns	(Min 0ns for
8212	0 ns	30 ns	synchronization D FF)

Table 5. Delay Assumptions

An exhaustive approach as Table 4 will more than pay itself back in terms of debugging the circuit. However, while this analysis may be helpful in understanding an existing circuit, it won't help as much in creating a new one. A general procedure for designing with memories is itemized below:

1. Determine how much processor time is available for memory access. Access from addresses is the most important parameter.
2. Determine how much buffering will be used (both to and from the memory) and how much delay there will be due to decode or qualifications in the circuit (in the memory design in Fig. 11, $\overline{WR}$ qualifies a write). Subtract these resulting delays from step 1 to get an effective access for the memory. If multiplexed address RAM is used go to 3, if not go to 4.
3. Determine how the $\overline{RAS}$ and $\overline{CAS}$ timings will be generated, be it one shots, delay lines, shift registers, etc. Adjust memory access available for the method chosen.
4. Select a memory that meets this criterion.
5. Design the system to meet all the specified parameters of the memory and verify.

Steps 1, 2 and 4 have been done for you in the Memory Compatibility Table for ROM, EPROM and Static RAM memories in a medium and minimum system. *Remember* - for dynamic RAM, Intel will soon be providing an 8202, a refresh, dynamic RAM controller that generates all $\overline{RAS}$, $\overline{CAS}$ control signals for a 64 kByte memory (made of 2117s).

Peripheral Compatibility - 3.125 and 5 MHz

Intel supports its processors with many LSI peripheral components that do a wide range of functions to simplify circuit design. The 8085A compatible peripherals have been denoted the "−5" notation to show compatibility. The "−5" notation also signifies that these devices are compatible with the 8085A-2 with one wait state interjected. This wait state is produced by taking the ready line low at the proper time as shown in Figure 19.

A list of these peripherals is shown in Table 6 with corresponding relevant specifications to illustrate 8085A-2 compatibility. The analysis for determining the resulting timings is similar to the analysis in the previous memory compatibility section.

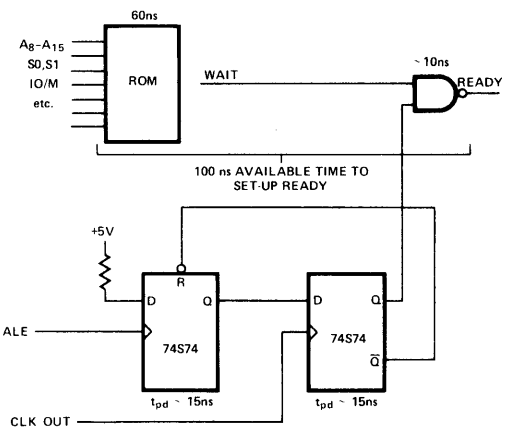


Figure 19. 8085A-2 Wait State Generator

Part No.	AC. Parameter	Min. (ns)	Max. (ns)	8085A-2 AC. Parameter	Margin vs. −2 Spec. (ns)
8251A	t_{RD}		200	t_{RD}	*150
	t_{RA} & t_{WA}	0		t_{CA}	60
	t_{DW}	150		t_{DW}	80
	t_{WD}	0		t_{WD}	60
	t_{RR} & t_{WW}	250		t_{CC}	*180
	t_{AR} & t_{AW}	0		t_{AC}	
8253-5	t_{RD}		200	t_{RD}	*150
	t_{RA}	5		t_{CA}	55
	t_{WA}	30		t_{CA}	60
	t_{DW}	250		t_{DW}	*180
	t_{WD}	30		t_{WD}	30
	t_{RR} & t_{WW}	300		t_{CC}	*130
	t_{RV}	1000		t_{RV}	**
	t_{AR} & t_{AW}	50		t_{AC}	65
8255A-5	t_{RD}		200	t_{RD}	*150
	t_{RA}	0		t_{CA}	60
	t_{WA}	20		t_{CA}	40
	t_{DW}	100		t_{DW}	130
	t_{WD}	30		t_{WD}	30
	t_{RR} & t_{WW}	300		t_{CC}	*130
	t_{RV}	850		t_{RV}	**
	t_{AR} & t_{AW}	0		t_{AC}	115
8257-5	t_{RD}		200	t_{RD}	*150
	t_{RA} & t_{WA}	0		t_{CA}	60
	t_{DW}	200		t_{DW}	30
	t_{WD}	0		t_{WD}	60
	t_{RR}	250		t_{CC}	*180
	t_{WW}	200		t_{CC}	30
	t_{AR}	0		t_{AC}	115
	t_{AW}	20		t_{AC}	95
8271 & 8273	t_{AD}		200	t_{AD}	*350
	t_{RD}		150	t_{RD}	*200
	t_{CA}	0		t_{CA}	60
	t_{DW}	150		t_{DW}	80
	t_{WD}	0		t_{WD}	80
	t_{RR} & t_{WW}	250		t_{CC}	*180
	t_{AC}	0		t_{AC}	115
8275	t_{RD}		200	t_{RD}	*150
	t_{RA} & t_{WA}	0		t_{CA}	60
	t_{DW}	150		t_{DW}	80
	t_{WD}	0		t_{WD}	60
	t_{RR} & t_{WW}	250		t_{CC}	*180
	t_{AP} & t_{AW}	0		t_{AC}	115
8279-5	t_{AD}		250	t_{AD}	300
	t_{RD}		150	t_{RD}	200
	t_{RA} & t_{WA}	0		t_{CA}	60
	t_{DW}	150		t_{DW}	80
	t_{WD}	0		t_{WD}	60
	t_{RR} & t_{WW}	250		t_{CC}	*180
	t_{RCY}	1000		t_{RV}	**
	t_{AP} & t_{AW}	0		t_{AC}	115

*With 1 "Wait State"
**Must allow for in Software

Table 6. Peripherals vs. 8085A-2

Taking note of asterisked margins shown on the comparison sheet: t_{AD}, t_{RD}, t_{RR} and t_{DW}, it is seen that they are all taken care of by introducing a wait state. The double asterisked margins deal with the t_{RV} spec on the 8255A-5, 8253-5 and 8279-5 peripherals. t_{RV} is the time from the rising edge of $\overline{WR}$ or $\overline{RD}$ to the next falling edge. To allow sufficient time for this spec it is necessary to delay the commands sent to these three peripherals. Enough dead time must occur to make up for the entire negative portion of the margin (for example: 790ns in the 8253-5 medium system). Since in the 8085A-2 every machine cycle is at least 200ns long, 4 machine cycles are sufficient time to allow peripheral control signal recovery (t_{RV}).

One may notice that all of the 8085A instructions take at least 4 T-states (providing a minimum of 800ns) giving ample time to meet this requirement, just by programming one instruction in between every command sent to the peripheral. I/O mapped I/O, which results in using the Input, Output instructions has this delay time built in when moving the data to be transferred into the accumulator. With memory mapped I/O, any instruction that accesses memory for data will provide the time necessary to not violate t_{RV} as a second fetch is performed.

Bus - Loading Considerations - Decoupling

For the cost conscious designer it is always helpful to know when buffering is needed and when it is not. How much can I load the 8085A output pins down? To answer this it is helpful to first list the DC requirements of the common types of logic loading and compare this to the capabilities of the 8085A.

	Maximum High-Level Input Current	Maximum Low-Level Input Current
TTL (single load)	40μA	1.6mA
Schottky or HTTL	40μA	2.0mA
MOS	10μA	10μA
LSTTL (single load)	20μA	400μA

The 8085A is capable of an IOL of 2mA (low) and IOH of $-$ 400μA. With this spec it is easy to come up with the possible combinations of D.C. loading that the designer can use without buffering:

LOADS	8085A, A-2 limiting factor (level)
1 TTL + 1 LSTTL	LOW
1 TTL + 36 MOS*	HIGH
1 SCHOTTKY or 1 HTTL	LOW
40 MOS (various combinations possible) *	HIGH
5 LS TTL	LOW

* Exceeds capacitive loading limit, to be discussed

If a user exceeds these DC loading limitations he must buffer that particular signal. Another factor that the designer **must** consider is the capacitive load that is seen by the 8085A outputs, which may very well be excessive even if DC loading is not. One may note that even though the 8085A can handle a DC load of 40 MOS devices or 36 MOS + 1 TTL, their collective input capacitances exceed the 150 pF max spec.

The timing specs of the 8085A are guaranteed as long as the 150 pF maximum loading is not exceeded, which includes the wires, components and parasitics. If the user exceeds this value and wants to guarantee his system timing he must either derate the system timings or use buffering.

What if you choose to ignore this limit and say you can live with the performance degradation? First the timing performance is not all that would degrade, a user must be willing to give up some reliability of his components (All MOS devices have this restraint). This is caused by the excessive switching currents that are needed for this extra loading capacitance. If reliability is not an important consideration, the user can load up to 300 pF on the 8085A bus, but the following correction factors must be used to adjust the timings:

for 150 pF $<$ 300 pF add .13 ns/pF

conversely if less than 150 pF:

for 25 $<$ CL $<$ 150 pF you can subtract .1/ns/pF.

What happens after 300 pF? If the user exceeds this, the noise levels become excessive and problems will result. How much is to much noise? 350 mvolts zero to peak. Prudent designers will always buffer when noise approaches this level, especially in the case of going from one board to another.

The above takes into consideration the actual specification considerations of when to buffer, but there are also transmission line and noise effects that must be considered. When working with dynamic RAMs small (20-30 ohm) resistors are commonly put in series in the address lines to help match impedance levels and reduce reflections. Note that this resistor should be chosen such that it does not severely degrade the voltage levels of the signal. Long parallel board traces with signals that could adversely affect each other should also be avoided to prevent cross talk problems.

By-passing is very important to prevent intermittent problems which often plague the board designer. Large bulk capacitors should be used at strategic locations on the board to prevent power supply droop. This becomes a major factor when there are many devices that can turn on at once and produce a considerable drain from the power supply (such as burst refresh in dynamic RAM).

To help smooth out the current spikes that naturally occur when devices turn on and off, it is recommended to liberally use small capacitors such as the monolithic and other ceramic capacitors which have low inherent inductance. Attached in the 2117 data sheet is a suggested layout of capacitors to effectively bypass the supply lines to ensure proper system operation. Cutting corners here will often times turn around and bite you.

Proper layout is an important consideration. Power supply lines should be well gridded to supply sufficient current to all areas of the board. A strong ground layout is advised to offset noise problems. Remember if the ground plane moves up in voltage because of excessive charge dumping in a particular area, the supply will drift up correspondingly. Sensing low levels often becomes an intermittent problem when proper ground is not provided.

APPLICATION EXAMPLE 1

MINIMUM SYSTEM APPLICATION AS A TEMPERATURE SENSOR

Overview

Following is an application example that illustrates the use of the interrupt and SOD pins on the 8085A, software for a block search routine, and the procedure for using and reading the 8155 counter. It is a simple application showing the use of the small but powerful 3-chip MCS-85 system as a temperature sensor (SDK-85 board used). This example can be modified to be an accurate industrial temperature controller, for several locations if desired.

The basic operation behind this application is a monostable multivibrator having its timing pulse duration controlled by a thermistor. The counter in the 8155 converts this timing pulse to a decimal count that is software mapped into a temperature and displayed in degrees C in the address field of the display in the SDK-85 Kit. For the purpose of keeping the software relatively simple, many approximations were incorporated into the code.

Detailed Hardware

The basic SDK kit was used for the initial hardware. This Kit provides for everything necessary to develop and debug a program through the use of the SDK-85 monitor, keyboard and display board. The kit provides for 256 bytes of RAM resident in the 8155 and 2K bytes of ROM or EPROM where the SDK-85 monitor is placed. (See the Intel SDK-85 User's Manual for copy of monitor software code.)

Figure 20 is a schematic of the SDK-85 Kit with only one 8155 and 8355. There is no buffering in this system as all components are on the same board and far below the maximum component loading. A monostable multivibrator (74121) is also shown with a thermistor connected to RE/CE.

The SOD output pin from the 8085A is used for the purpose of starting the monostable multivibrator in generating its temperature controlled timing pulse. This pulse is created by the RC time constant provided for by the thermistor acting as a variable resistor and a .1μF capacitor to put the timing pulse in the desired timing range.

The inverted output of the monostable multivibrator (one shot) has been directly connected to the RST 6.5 pin on the 8085A. Since this pin is high level sensitive, it is necessary to disable interrupts in the program until after the pulse from the one shot goes low.

The hardware addressing in the configuration shown allows for several code spaces that could be used. The RST and TRAP interrupt lines on the 8085A also have hardware start addresses but many of these are altered by the SDK monitor. Table 7 should be useful in understanding the addresses used in the software that follows. Each memory/ I/O component in the basic SDK-85 system is enabled by a signal coming from the 8205 address decoder. Since no expansion chips are used, output enables 00 (8355 monitor ROM), 03 (8279 Keyboard) and 04 (8155 RAM) were the only ones needed. Additional memory and/or I/O could have been incorporated using other output enables from the 8205.

Memory/ I/O Device	Function	Output from 8205	code space
8155	RAM space	04	2000 - 20FF (20 - 20FF are reserved for monitor RAM locations)
8355	ROM space	00	0000 - 07FF
8279	Keyboard/display controller	03	1800 - 1FFF

stack pointer

Since the monitor uses locations 10C8 through 20FF, the stack pointer must be initialized to 20C8 or less.

	8085A jump address	Usage	monitor mapped address
trap	24H	$\overline{T0}$ of 8155	0157
RST 5.5	2CH	8279 interrupt	028E
RST 6.5	34H	oneshot interrupt	20CE
RST 7.5	3CH	vector interrupt	

I/O ports address	Function
00	Monitor ROM Port A (8355)
01	Monitor ROM Port B (8355)
02	Monitor ROM Port A (8355) Data direction register
03	Monitor ROM Port B (8355) Data direction register
20	Basic command/status register
21	Basic RAM Port A
22	Basic RAM Port B
23	Basic RAM Port C
24	Basic RAM LOW order byte of timer count
25	Basic RAM HIGH order byte of timer count

Table 7. Addressing

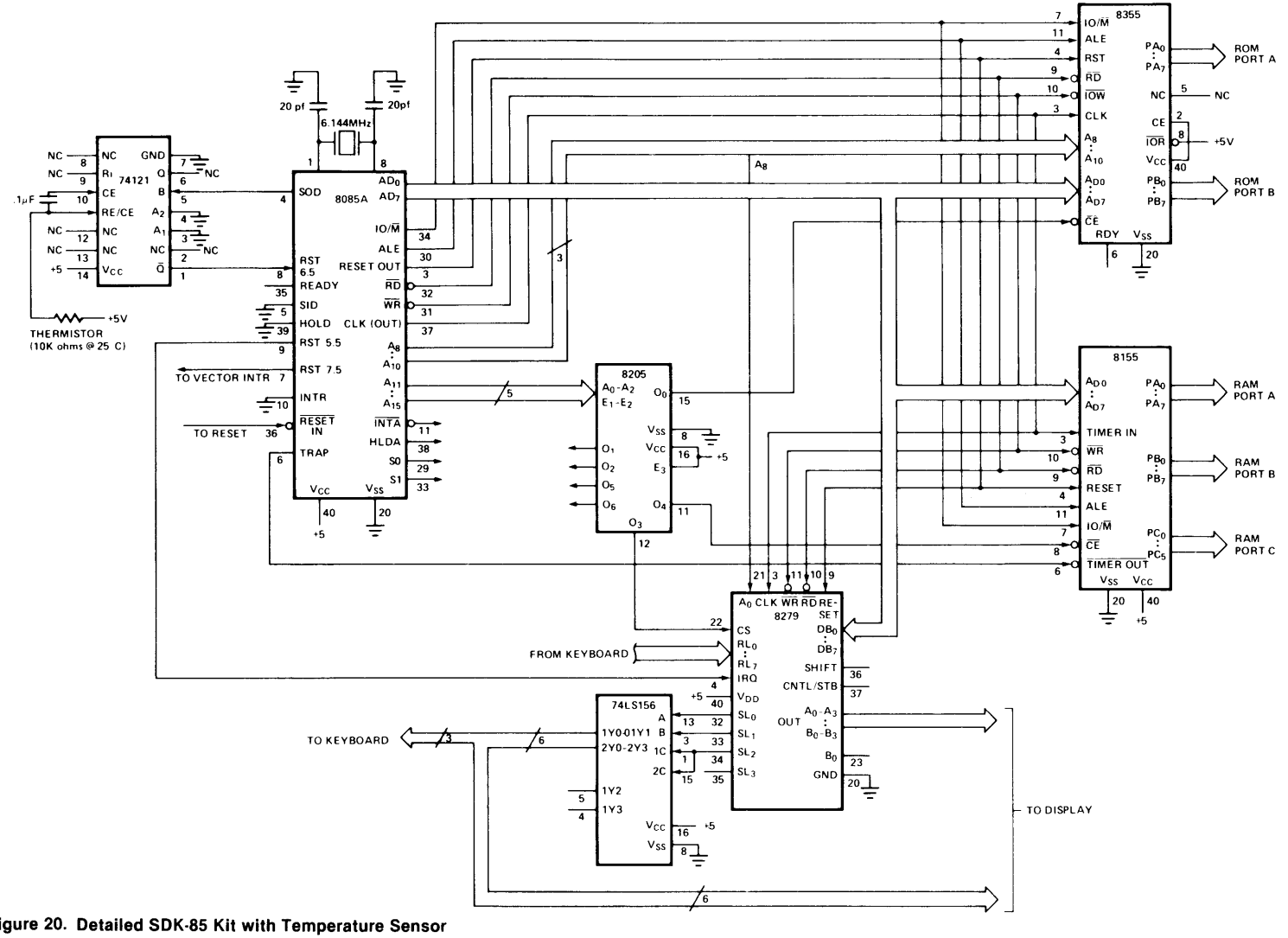


Figure 20. Detailed SDK-85 Kit with Temperature Sensor

Software

The software (at end of section) for this application illustrates several features of the 8085A, such as the programming of the SOD line, interrupts and 8155 counter. Additionally, an example of a block search routine is illustrated.

Figure 21 is a flow diagram of the program. It has been cross referenced with program lines to the actual software for the reader's convenience. Following through the flow diagram it is seen that the interrupts are disabled in the beginning as the one shot is outputting a high level on its Q output and interrupt pin 6.5 is high level sensitive. However, this high level will not be recognized until the level goes low and then high again. If the user would prefer a positive pulse interrupt the 8085A a dual one shot can be used with one triggering the other, or just a simple inverter. Starting and loading the counter is as described in the 8155 data sheet with the Port addresses being given in the previous Table (7). Code lines 18-23 represent placing the counter in the counter mode (single terminal count pulse at the end of count) and starting the count, having the count clocked by the 8085A clock out pin. Reading the counter is not as straight forward and will be approached shortly. Code lines 28-32 are representative of programming the SOD line to output a pulse. This pin is intended for serial I/O interfaces such as a teletype, but as seen in this application, it can also be used as a single I/O port.

After the pulse is presented to the one shot, the interrupts enabled, the processor idles (lines 36, 37; Halt could have just as easily been used) until interrupted. Through the design of this application it was known that the down counter would never reach terminal count, as it is only being used as a pulse to digital count converter.

To read in the count value it is best that the counter is first stopped. The least and most significant bytes of the count length register in the 8155 are read using the same port addresses as was used during loading the counter, as seen in code lines 42-47. If one looks at this value and knows how many pulses occurred, he would come to the conclusion that there is a gross discrepancy! The reason for this is that the counter in the 8155/6 was designed to make its square wave function generation easy and when used in the counter mode, it counts by two's. For this application (where length of time is mapped into a temperature) and other similar event timing applications it is imperative to have an intelligible count returned from the 8155.

The counter in the 8155 is essentially a count down by 2 counter. After it counts down by 2 the initial value loaded by the user, it reloads the initial count (initial count −1 if odd) and counts down by 2 again until terminal count is reached. When reading the counter, the least significant bit of the counter does not represent the least significant bit of the count, but which half of the countdown operation you are in. If this bit equals 1, the 8155/6 counter is counting down by 2 in the first half, and if it is zero you are in the second half of the operation. Because of this method of down counting there are two restrictions placed on its use:

1. The user can not use the initial value of 1 to detect only one pulse.
2. The user can not discern (through reading the counter) whether exactly one or two pulses on the timer input pin has occurred if he loaded in an initial odd count (does not apply to even). After three pulses the user can determine exactly how many pulses occurred. Note that this restriction only applies to reading the counter, the $\overline{TO}$ pin pulses correctly after the correct number of pulses regardless of what is read from the counter.

The first pulse to the 8155/6 counter (high level sensitive) loads the count length register, which says that the counter is not readable until a pulse occurs. If the user tries to read before a pulse is provided he will read a previous or old value. Now what is done with the value read?

Good question. An adjustment routine to convert this value read to an actual count can be summarized as follows:

1. Read in 16 bit count length register.
2. Reset the upper two bits (mode bits).
3. Reset carry and rotate right all 16 bits through carry.
4. If carry is set add 1/2 of full original count (1/2 (full count −1) if full count is odd).

In the software for this application is a general purpose routine to do this; lines 179-199. To call this routine it is assumed that the lower order byte of the counter is in register C, higher order byte in register B and full original count is in HL. Contents of H, L, B and C are destroyed returning actual count in BC register pair. To obtain the number of pulses that occurred, subtract this number from full original count and add 1.

Converting this remaining count to an actual temperature can be done by various methods but it was chosen to do a software map through the use of a block search routine. Table 8 presents approximations of what the remaining count should be for each temperature. To keep the software simple it was only necessary to compare the most significant byte to a list to find the appropriate temperature. This search routine is set up to find a "less than" match, incrementing the HL register as a pointer when a compare is made. The code for this search routine is in lines 118-144 and is optimized to be a fast 8 byte block search. This search routine can be made to search for a match by replacing all return on carry with return on zero. The performance of this subroutine is as follows:

Byte time = (11 + (166/8) N) CC/N = (11/N + 20.8) CC
where: CC = microseconds per clock cycle
N = total number of bytes searched
Byte time = time per byte searched