Unit – 3: Chapter - 7

Interfacing Semiconductor Memories and I/O Devices

Topics Covered

1. Types of Main Memory – ROM
2. Types of Main Memory – RAM
3. CPU Read/Write Timing
4. Address Decoding Techniques
5. Interfacing Dynamic RAM

Memory that interfaces directly to the microprocessor is referred to as ***main memory***.

It has the following characteristics:

1. Any location can be accessed at random.
2. Each byte has its own unique address.
3. Data is read or written in one CPU bus cycle.

The best known examples of main memory are RAM and ROM.

The basic RAM cell is a flip-flop that can be set or reset.

It is this feature of RAM that allows it to be written as well as read.

RAMs are said to be a ***volatile memory*** type.

Any information stored in RAM must ultimately be lost.

For this reason, a more permanent type of memory is required to save important data and program files.

ROM is a ***nonvolatile*** main memory type that can only be read.

Because of their nonvolatility, ROMs are often mapped to cover the CPU's reset address.

In this way they can be used to "boot" the computer up from a "cold" start.

1. Types of Main Memory - ROM

1. Mask-Programmable ROMs

A ROM is made up of an address decoder, a programmable memory array, and a set of output buffers.

In the figure below, n = 4 and m = 5.

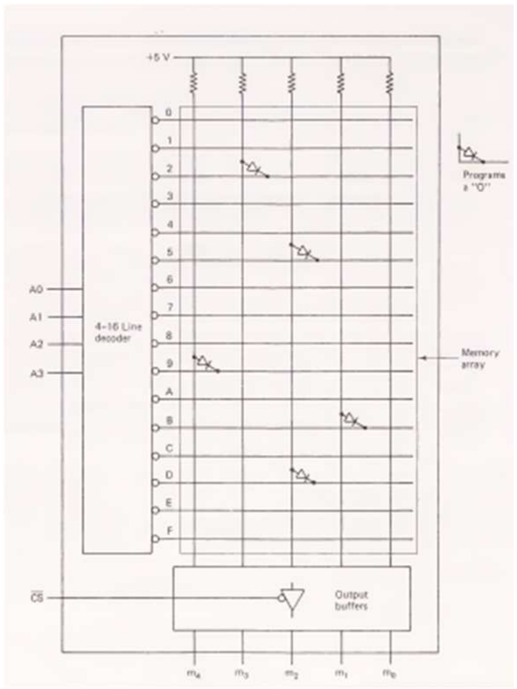
When a 4-bit address is applied to the ROM, one of the 16 row lines will go low.

A diode connected between a row line and a column line will program that output bit low.

The absence of a diode will program a logic 1.

A ROM provides one m-bit output word for each possible input combination.

A mask-programmable ROM is one in which the diode connections are programmed at the factory according to a truth table supplied by the user.



2. Field-Programmable ROMs

There are several types of ROMs that can be programmed by the user in the field.

These devices are referred to as PROMs or programmable read-only memories.

Fusible-Link PROMs	This type of PROM uses a low-current fusible link in series with the output. By applying a current pulse to the desired output, the fuse can be melted and a logic 1 or 0 permanently programmed. These devices have a typical access time of 35 ns. A typical application would be a “boot” PROM in a microcomputer system.
UV-Light-Erasable PROMs	These are the most popular type of PROM used in microcomputer systems. This is the erasable programmable read-only memory (EPROM). This device can be programmed, erased, and reprogrammed many times over by the user.

2. Field-Programmable ROMs (Contd …)

There are several disadvantages to the UV EPROM. These are:

1. The device must be removed from the circuit board to be erased.
2. Byte erasure is not possible; all cells are erased when exposed to UV light.
3. The quartz window package is expensive.

Because of these problems much research has been devoted to developing an electrically erasable nonvolatile memory device.

The result is the E^{2}PROM.

3. Electrically Erasable PROMs (E^2PROMs)

This device can be programmed and erased without removing the chip from its socket.

In addition, both byte and bulk erasure modes are possible.

2. Types of Main Memory - RAM

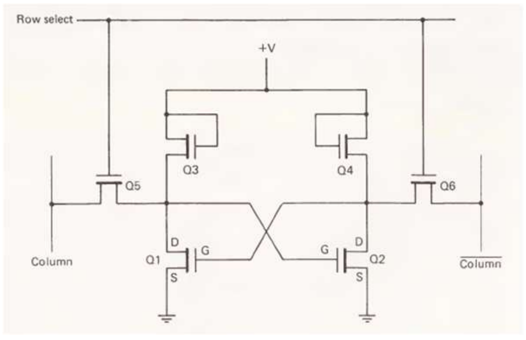
1. Static RAM

In the static RAM (SRAM), four to six transistors are connected to form a simple RS flip-flop.

The standard six-transistor static memory cell is shown in the figure below.

Q1 and Q2 are the active devices, while Q3 and Q4 are biased as resistive loads.

Transistors Q5 and Q6 are used to gate data onto the internal data bus.



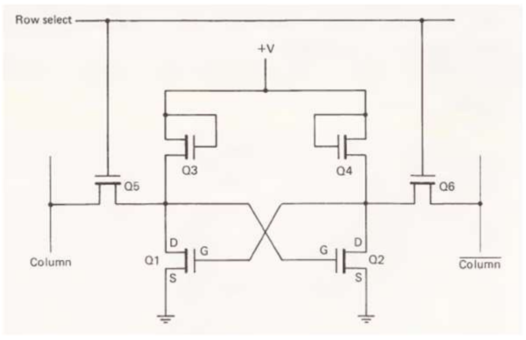
1. Static RAM (Contd …)

As with all flip-flops, the latching mechanism is due to the cross-coupling of the active devices, Q1 and Q2. If the Q1 drain is at 0 V, this level will hold Q2 OFF and its drain terminal will be high.

This high level, in turn, holds Q1 ON and its drain low.

Data is written to the cell by applying the desired data (and its complement) to the column and column lines with Q5 and Q6 ON.

Data is read out by enabling Q5 and Q6 and reading the column line.



1. Static RAM (Contd …)

This static cell is volatile because it will lose its data if power is lost. Each cell will also power-on in an unpredictable state.

Static memories are best known for their ease of interfacing and fast access times.

2. Dynamic RAMs

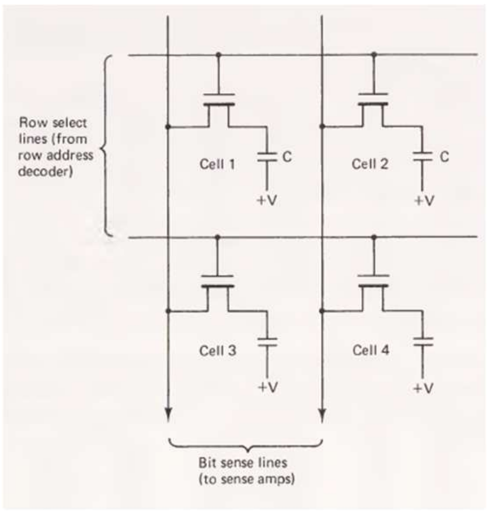
One of the goals of main memory technology is to produce a high-bit-density component.

For this, the six transistors per cell may not be the best choice.

Dynamic RAMs or DRAMs are based on the well-known capacitive input nature of an MOS transistor.

In this scheme the basic storage cell is shrunk to a single bit-select transistor and storage capacitor.

A portion of a dynamic RAM memory array is shown in the figure below.


Row select lines (from row address decoder)
Cell 1
C
+V
Cell 2
C
+V
Cell 3
+V
Cell 4
+V
Bit sense lines (to sense amps)

2. Dynamic RAMs (Contd …)

For a given integrated-circuit die size, a DRAM will be four to six times as dense as the corresponding static die.

For example, when the first 16K-bit static RAMs were being announced, 64K-bit dynamic RAMs were becoming available.

In addition to this density improvement, DRAMs generally require much less power than do their static counterparts.

2. Dynamic RAMs (Contd …)

The latest trend is to fabricate DRAMs using CMOS technology versus the older NMOS technology.

CMOS is easier to design.

It consumes much less power, and provides faster access times.

The DRAM's greatest shortcoming is that the storage node is not perfect: it leaks.

This necessitates a refresh operation—that is, sense the charge, amplify it, and then rewrite it.

This must be performed once every 2 ms to each cell in the memory.

Memory Organization

From the manufacturer's standpoint, the most important number describing a memory component is the total bit capacity.

But from a user's standpoint, it is just as important to know how this memory is organized.

The organization of a memory refers to the number of bits in the output word.

Most DRAMs are just 1 bit wide, to accommodate a maximum number of address lines.

The advantage of the byte-wide devices is that a single chip can be used to provide a practical memory in a small system.

This allows miniaturization of the overall hardware.

3. 8086 CPU Read/Write Timing

To a “hardware type” a microprocessor is a complex timing unit.

There is little we can do to change the CPU timing other than add WAIT states.

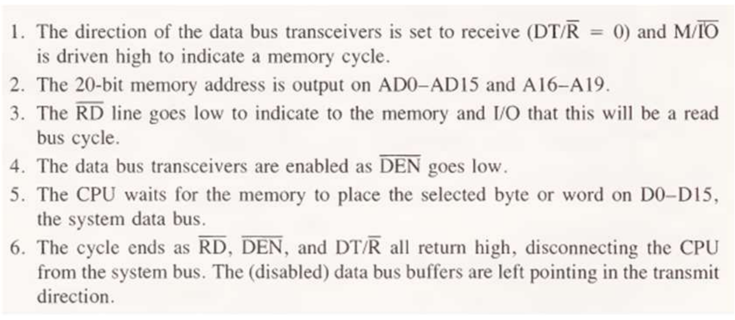
Our goal will be to come up with a set of timing “windows” (or frameworks) for transferring data between the CPU and main memory.

Minimum-Mode Read/Write Timing Specifications

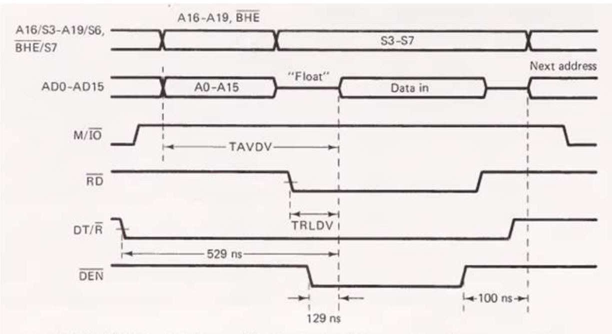
Memory read timing (CPU read from memory):

The following figure shows the signals involved in a minimum-mode read cycle.

The sequence refers to the CPU reads a byte or a word from memory.



1. The direction of the data bus transceivers is set to receive ($DT/\overline{R} = 0$) and $M/\overline{IO}$ is driven high to indicate a memory cycle.
2. The 20-bit memory address is output on AD0–AD15 and A16–A19.
3. The $\overline{RD}$ line goes low to indicate to the memory and I/O that this will be a read bus cycle.
4. The data bus transceivers are enabled as $\overline{DEN}$ goes low.
5. The CPU waits for the memory to place the selected byte or word on D0–D15, the system data bus.
6. The cycle ends as $\overline{RD}$, $\overline{DEN}$, and $DT/\overline{R}$ all return high, disconnecting the CPU from the system bus. The (disabled) data bus buffers are left pointing in the transmit direction.



TAVDV = 3TCLCL − TCLAVmax − TDVCLmin = 3 × 200 ns − 110 ns − 30 ns = 460 ns (8086)
TRLDV = 2TCLCL − TCLRLmax − TDVCLmin = 2 × 200 ns − 165 ns − 30 ns = 205 ns (8086)

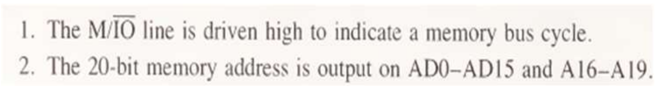
There are two important timing windows provided by the 8086.

Address access time	This is also referred as the access time. It is the time from output of the memory address on AD0-AD15 and A16-A19 until valid data is expected by the CPU. This is called TAVDV in Intel's literature. This is calculated to be 460ns minimum.
Read access time	This is the time from RD active (low) until valid data is expected by the CPU. This is called TRLDV in Intel's literature. It is 205ns minimum for a 5-MHz 8086 minimum-mode CPU.

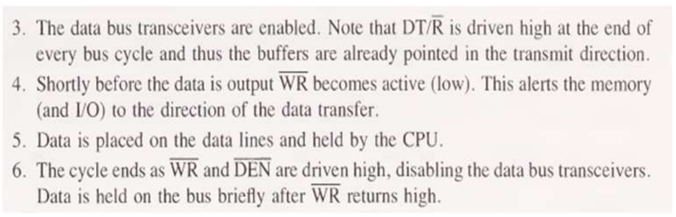
Minimum-Mode Read/Write Timing Specifications

Memory write timing (CPU write to memory):

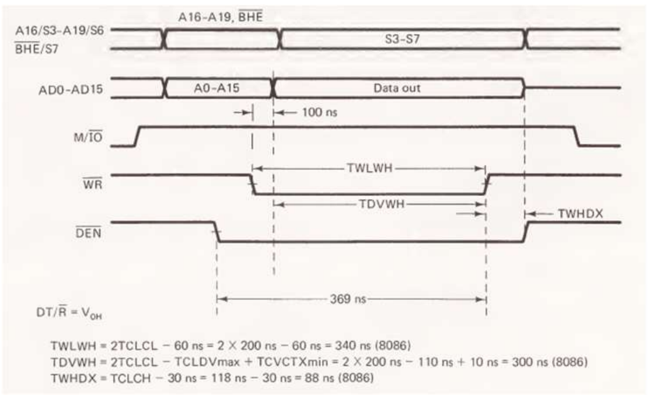
The following figure shows the timing for a minimum-mode write cycle, for which the following sequence occurs:



1. The $M/\overline{IO}$ line is driven high to indicate a memory bus cycle.
2. The 20-bit memory address is output on AD0–AD15 and A16–A19.



3. The data bus transceivers are enabled. Note that $DT/\overline{R}$ is driven high at the end of every bus cycle and thus the buffers are already pointed in the transmit direction.
4. Shortly before the data is output $\overline{WR}$ becomes active (low). This alerts the memory (and I/O) to the direction of the data transfer.
5. Data is placed on the data lines and held by the CPU.
6. The cycle ends as $\overline{WR}$ and $\overline{DEN}$ are driven high, disabling the data bus transceivers. Data is held on the bus briefly after $\overline{WR}$ returns high.



TWLWH = 2TCLCL − 60 ns = 2 X 200 ns − 60 ns = 340 ns (8086)
TDVWH = 2TCLCL − TCLDVmax + TCVCTXmin = 2 X 200 ns − 110 ns + 10 ns = 300 ns (8086)
TWHDX = TCLCH − 30 ns = 118 ns − 30 ns = 88 ns (8086)

4. Address Decoding Techniques

The 8086 microprocessor provides a 20-bit memory address that allows up to 1MB of main memory.

However, most memory interfaces do not fill this entire range. That is, for a given memory design, several of the address lines are going to be “unused.”

However, these unused lines are very important because they determine the range of addresses the memory interface will occupy.

To examine the extra address lines and enable the memory for a specified range of addresses an address decoder is used.

This is an important aspect of any memory design, as one block of memory must not be allowed to overlap another.

Full Decoding

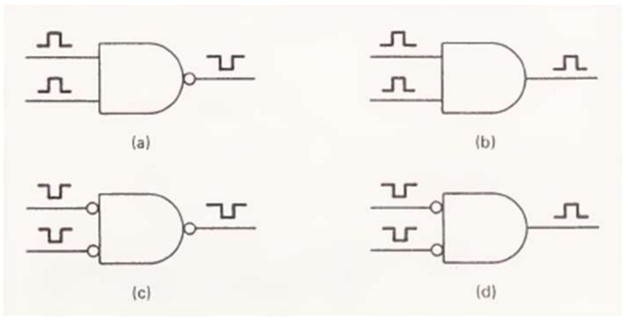
A digital decoder is a circuit that recognizes a particular binary pattern on its input lines and produces an active output indication.

For example, a two-input NAND gate recognizes the input pattern “11” and produces an active-low output only when this input combination is applied.

A two-input AND provides a similar function but produces an active- high output indication.

OR and NOR gates can also be used as decoders when their alternate logic symbols are used.

The four decoder combinations are shown below.



(a) NAND (b) AND (c) OR (d) NOR

Partial Decoding

The smaller the memory block, the more address lines there will be to decode.

All of this decoding circuitry can become meaningless if much of the memory space is unimplemented.

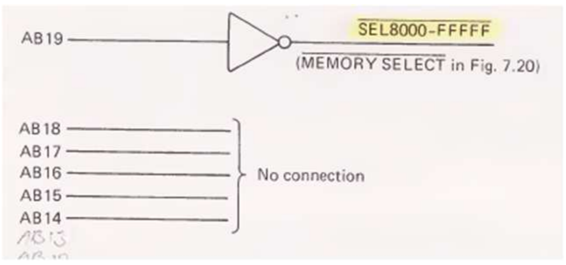
In such systems partial decoding is often used.

This is a technique in which some of the decoded address lines are simply ignored.

The following figure gives an example for the 16K-byte EPROM interface.

In this circuit, the decoder has been shrunk to a single inverter, with address lines AB18 to AB14 left untested.

In effect, these lines become "don't cares."



Partial Decoding (Contd …)

Partial decoding is often used in microcontroller applications where the microprocessor is embedded into the final product.

In these applications, the memory configuration is fixed and the ability to expand the memory is not required.

Block Decoding

In a practical microcomputer the memory array often consists of several blocks of memory chips.

This will require a separate decoder for each memory block unless a special block decoder is built.

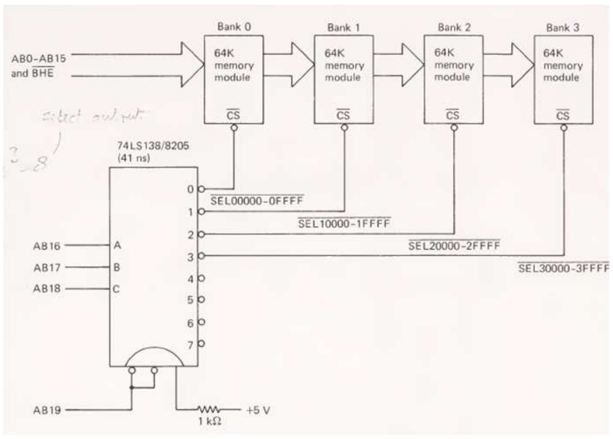
Block Decoding (Contd …)

The following figure shows a 256K-byte memory array.

It consists of four 64K-byte blocks of memory and a 74LS138 (Intel part number 8205) 3-to-8-line decoder.

The low-order address lines AB0-AB15 select one of 64K bytes in each memory bank.

However, the decoder allows only one bank to be enabled at a time.


AB0–AB15 and $\overline{BHE}$
Bank 0
Bank 1
Bank 2
Bank 3
64K memory module
$\overline{CS}$
select output
3-8
74LS138/8205 (41 ns)
AB16
AB17
AB18
A
B
C
0
1
2
3
4
5
6
7
$\overline{SEL00000-0FFFF}$
$\overline{SEL10000-1FFFF}$
$\overline{SEL20000-2FFFF}$
$\overline{SEL30000-3FFFF}$
AB19
1 kΩ
+5 V

5. Interfacing Dynamic RAM (DRAM)

In 8086, the static memory (SRAM) is filled only about 3% of the memory space.

Because of the low bit density, SRAM chips are normally used only in systems requiring a small amount of RAM (<32K bytes) or very fast access times.

These requirements would be met by a microcontroller-based system or a computer designed for high-speed data acquisition and analysis, for example.

It is common to find the 8086 supported by 128K to 512K bytes of dynamic RAM memory.

The remaining memory space may be filled with a bootstrap ROM and additional RAM chips serving as the video screen memory.

Dynamic RAM Timing

Packaging

To be consistent with the dynamic RAM's high bit density, most DRAM chips are packaged in standard 300-mil-wide 16-pin DIPs. (15mil = 0.381mm)

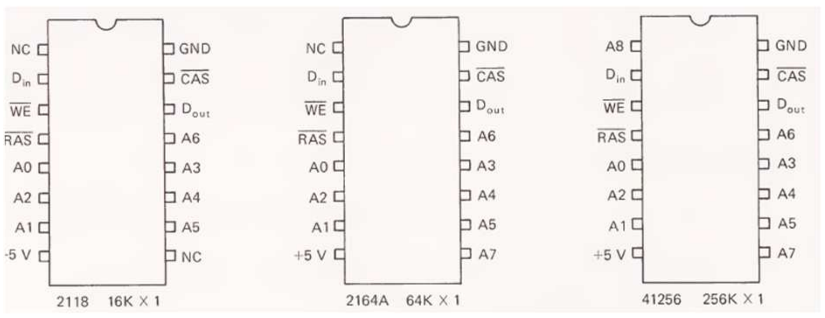
This allows a very dense memory to be constructed in a minimum amount of PCB (printed circuit board) area.

However, a 16-pin package does not have enough pins for all the address lines of a 256K memory part.

For this reason, DRAMs are organized 1 bit wide with multiplexed address pins.

Dynamic RAM Timing (Contd …)

The industry standard pinning for 16-pin DIP 16K, 64K, and 256K DRAMs is shown below.



NC: Not Connected

It is interesting to note that the 16K and 64K chips actually have unused pins.

By adopting this standard pinning arrangement, designers are able to design memory interfaces that can easily be upgraded.

Although the address pins are labeled AO to A6 (or A7 or A8), the high-order address lines, A7 to A14 (or A15 or A16), are also applied to these pins via a multiplexing scheme.

Unit – 3: Chapter - 8

I/O Devices

Topics Covered

1. Parallel I/O
2. Serial I/O
3. Programmed I/O
4. Interrupt Driven I/O
5. Direct Memory Access

At the most basic level there are two types of I/O ports: parallel and serial.

Because the microprocessor naturally works with data in 8- or 16-bit "chunks," the parallel port is the easiest to implement.

All bits comprising the data word are input or output together in parallel.

In a serial I/O port, the data bits are lined up and transmitted one bit at a time.

It should be apparent that this technique will be slower than the parallel port design, but it does have certain advantages.

The serial data bits can also be converted to audio tones (using a circuit called a modem) and transmitted over the telephone network. A receiving modem converts the tones back into digital I's and 0's.

In this way one computer can communicate with another thousands of miles away.

Regardless of the I/O port design (parallel or serial), the microprocessor must be synchronized to the speed of the peripheral.

Some peripherals, like printers and plotters, cannot accept data as fast as the microprocessor would like to output it.

On the other hand, floppy-disk drives and Winchester disks may require data faster than the processor can supply it.

Both cases must be handled carefully to prevent a loss of data.

1. Parallel I/O

The hardware requirements for a parallel I/O port are similar to those of a RAM or ROM interface.

When the CPU performs an output instruction (I/O write cycle), the data on the bus must be stored by the port.

Similarly, when an input instruction is executed (I/O read cycle), the I/O port must gate its data onto the data bus lines.

Just as each memory location has its own (memory) address, each I/O port has its own (port) address.

I/O Bus Cycle Timing

The 8086 and 8088 have only two I/O instructions:

1) IN AL (or AX), port

2) OUT port, AL (or AX)

Direct	IN AL (or AX), port OUT port, AL (or AX)	Here, the I/O port address is supplied within the instruction and restricts the access to ports with addresses between 0 and 255.
Indirect	IN AL (or AX), DX OUT DX, AL (or AX)	It uses register DX to hold the port address. This allows access to full range of I/O ports from 0 to 65,535.

The advantage of the indirect form is that an I/O procedure can be set up and shared between several peripherals by passing the port address (in register DX) to the procedure.

Designing a Parallel Input Port

The following figure shows a 16-bit input port.

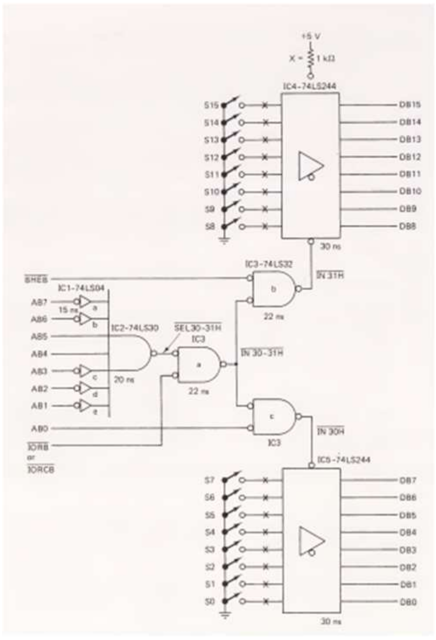
Sixteen switches are used to simulate the input data.

IC1 and IC2 decode the (direct) port address on AB1-AB7.

The output of IC2 is the port select signal SEL 30- 31H.

IC3a combines this signal with IORB to generate the device select pulse signal (DSP) IN 30-31H.

This signal is so called because it will be active only for input instructions that select the devices wired to ports 30H or 31H.



Memory-Mapped I/O

The address space of the 8086 is divided into 1,048,576 bytes of memory space and 65,536 bytes of I/O space.

These two regions do not overlap because memory addresses are selected with the memory commands (MEMR, MRDC, MEMW, MWTC), while the I/O addresses are selected with the I/O commands (IOR, IORC, 10W, IOWC).

Consider designing a one-byte (or word) read/write memory.

We would use latches (or flip-flops) to store the data written during a memory write cycle, and tri-state gates to drive the bus during a memory read cycle.

This is the essence of memory-mapped I/O.

The advantage of memory-mapped I/O is the availability of large number of instructions and addressing modes for memory referencing.

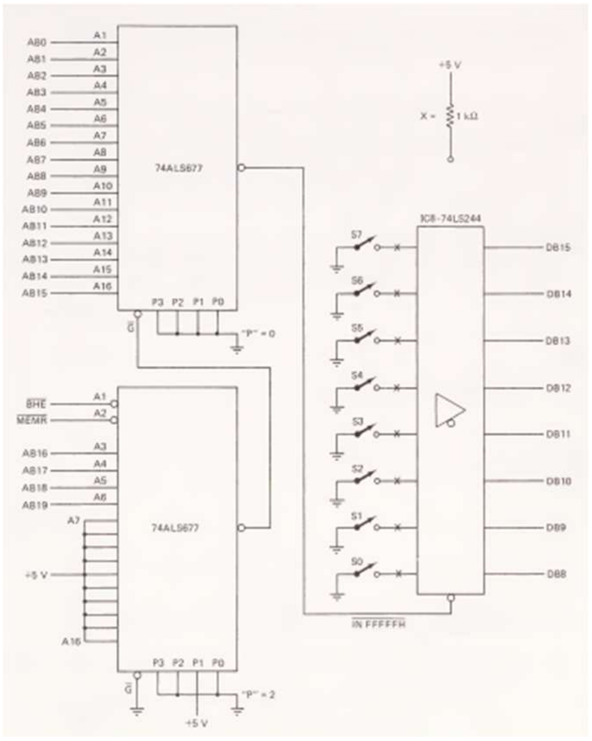
Memory-Mapped I/O (Contd …)

The following figure shows an 8-bit input port memory-mapped to address FFFFFH.

Two 74ALS677 address comparators are required to decode all 20 address lines and two control signals.

Comparators are 1-bit digital circuits made up from standard AND, OR and NOT gates. These are used to compare binary digits.

Memory-mapped I/O is most appropriate when the peripheral to be interfaced requires a large block of consecutive addresses.



2. Serial I/O

Asynchronous Serial Communications

One of the most common applications for a serial I/O port is to interface the keyboard on a **video display terminal (VDT)**.

In this circuit, each keystroke generates a 7-bit ASCII code which is converted to bit-by-bit serial and then transmitted to a computer over a two- or three-conductor cable.

Because even the fastest typist cannot exceed data rates of 60 to 100 words per minute, it is a good match for the (relatively) slow transmission rate of the serial port.

An important characteristic of this interface is that, at some times, the serial port will be required to transfer data at 10 to 20 characters/s, but at other times the data rate may be only 1 or 2 characters/s.

Indeed, most of the time the keyboard is not in use and the data rate is zero.

Because of this uncertain data rate, an asynchronous communications protocol must be established.

Start bits, stop bits, and the baud rate

The accepted technique for asynchronous serial communications is to hold the serial output line at a logic 1 level (a “mark”) until data is to be transmitted.

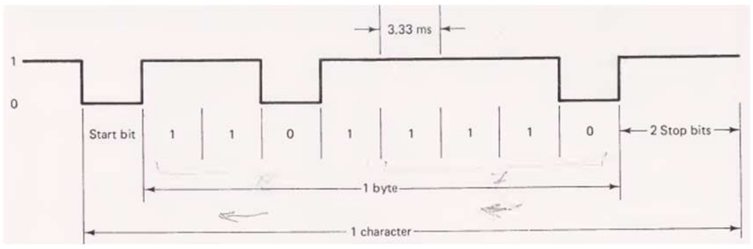
Each character is required to begin with a logic 0 (a “space”) for one bit time.

This first bit is called the start bit and is used to synchronize the transmitter and receiver.

Start bits, stop bits, and the baud rate (Contd …)

The following figure illustrates how the data byte 7BH would look when transmitted in the asynchronous serial format.

The data is sent least significant bit first and framed between a start bit (always a 0) and one or two stop bits (always a 1).



The start and stop bits carry no information but are required because of the asynchronous nature of the data.

The data rate can be expressed as bits/s or characters/s.

The term bits/s is also called the ***baud rate .***

Generating and recovering asynchronous serial data

All microprocessors are capable of generating serial data without special hardware.

Each bit to be transmitted is rotated to the bit 0 position of the accumulator and output.

The DELAY procedure determines the baud rate.

Standard asynchronous serial communications protocols

Protocols define certain rules that should be followed to help standardize the communications technique.

An example is the adoption of a logic 0 for a start bit and a logic 1 for a stop bit.

When setting up a serial port several parameters must be specified. The most common are:

1. Data bits/character, usually 5 to 8
2. Stop bits, one or two
3. Parity bit, used to detect single bit errors, may be specified as odd or even or no parity
4. Baud rate

Synchronous Serial Communications

The start and stop bits of asynchronous serial data represent wasted overhead bytes that reduce the overall character rate no matter what the baud rate.

Even adding a parity bit can reduce the transfer rate by 10%.

But giving up the start and stop bits will require some means of synchronizing the data.

How will we know when the data starts and when to sample it?

Here, we examine two common synchronous serial protocols that answer these questions.

1. Bisync protocol

As there is no start bit, a special sync character is required in all synchronous serial formats.

This character tells the receiver that data is about to follow. The USART, accordingly, must have a special "hunt" or "search" mode so that the sync character can be found.

As there is no stop bit, a clock signal usually accompanies the synchronous data to maintain synchronization.

When synchronous serial data is to be transmitted over the telephone network, it is not possible to provide a separate clock channel.

In this case, a special synchronous modem is used that encodes the data and clock into a single signal. The receiving modem separates the data and clock signals.

Another difference when compared to asynchronous serial is that the clock rate is the same as the baud rate.

1. Bisync protocol (Contd …)

In the bisync protocol, several special (ASCII) characters are used to control the data transfer as shown in the following table.

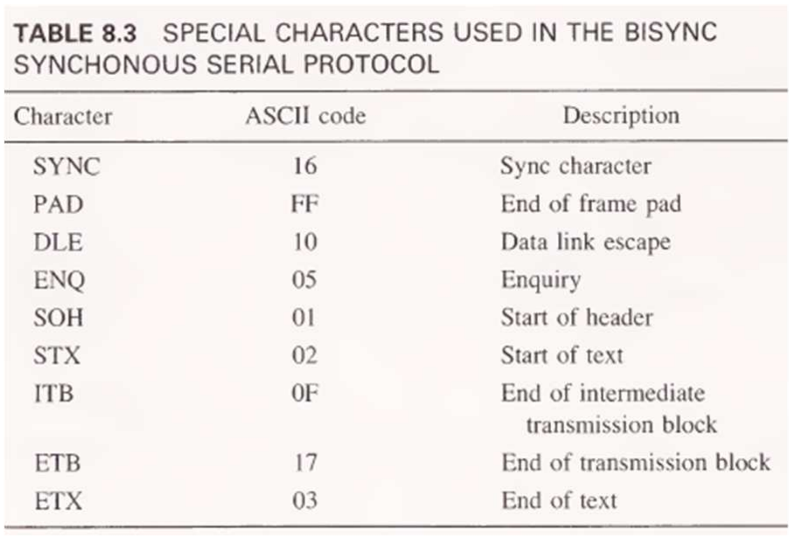
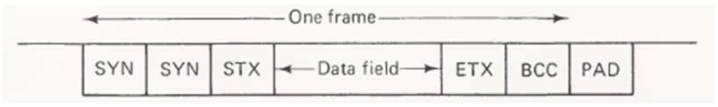


TABLE 8.3 SPECIAL CHARACTERS USED IN THE BISYNC SYNCHONOUS SERIAL PROTOCOL

Character	ASCII code	Description
SYNC	16	Sync character
PAD	FF	End of frame pad
DLE	10	Data link escape
ENQ	05	Enquiry
SOH	01	Start of header
STX	02	Start of text
ITB	0F	End of intermediate transmission block
ETB	17	End of transmission block
ETX	03	End of text

1. Bisync protocol (Contd …)

The following figure illustrates one “frame” of a synchronous message.



The synchronous data is framed between special control codes.

In the above frame, two sync characters are output followed by the start of text (STX).

The data bytes follow. This block may consist of 100 or more data bytes or simply be other control codes.

ETX signifies end of text.

BCC is a block check character used for error detection.

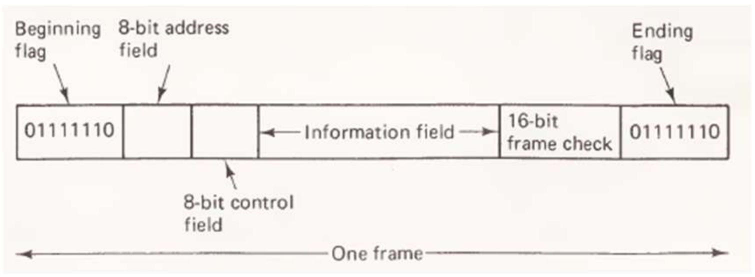
PAD is the character output when no data is being transmitted and corresponds to the “mark” output in asynchronous serial.

2. Serial data link control (SDLC)

This format was developed by IBM for use with their Systems Network Architecture (SNA) communications package.

The following figure illustrates one frame of data using this protocol.

It is similar to bisync but is not byte oriented.



2. Serial data link control (SDLC) (Contd …)

The SDLC receiver searches for the beginning flag (01111110) as its sync character.

An 8-bit address field follows, allowing each frame to be addressed to a particular station among a network of stations.

Control characters are identified by a sequence of six or more logic l’s.

The information field can be of any format.

The transmitter will automatically insert 0’s in this field if five or more logic l’s should appear in sequence.

This will avoid inadvertent control characters appearing in the information field.

The receiver automatically deletes these 0’s.

The 16-bit frame check is used for error detection, similar to the BCC character in bisync.

The frame ends with the ending flag.

3. Programmed I/O

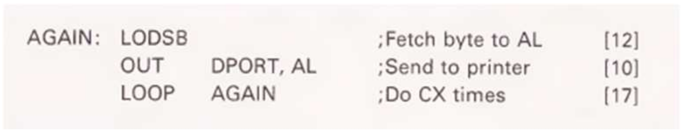
Regardless of the type of I/O port, serial or parallel, a strategy must be developed to control or synchronize the flow of data through that port.

Example

Assume that a 100-character/s (cps) printer is interfaced to a parallel output port.

This printer is capable of printing a new character every 10 ms (1/100 cps).

Consider the following 8086 output routine, which might be used to supply data to the printer.



```
AGAIN:  LODSB               ;Fetch byte to AL     [12]
        OUT    DPORT, AL    ;Send to printer      [10]
        LOOP   AGAIN        ;Do CX times          [17]
```

The numbers in brackets represent the number of T states required for each instruction.

T-states refers to the portion of an operation carried out in one clock period.

Using the above routine, the 8086 can output a new byte to the printer every 39 T-states.

At 5 MHz this is a new character every 7.8 ms and corresponds to 1,28,205 characters/s!

There is an obvious mismatch in the data rates of the printer and microprocessor.

Data Transfer Rate

The 8086 is capable of interfacing faster peripherals.

We can calculate the maximum rate at which the program can transfer data by assuming the peripheral will be ready after only one loop through the polling instructions.

In this example 30 T states are required to test the BUSY flag, and 49 T states to fetch the data byte, advance the pointer, generate the strobe pulse, and test for done.

In all, 79 T states are required or 15.8 ms at 5 MHz. This corresponds to a data transfer rate of 63,291 characters/s.

To put this number into perspective, an 8-inch double-density disk drive reads and writes data to a floppy disk at 62,500 bytes/s.

Thus the 8086 might be fast enough to interface this peripheral.

4. Interrupt Driven I/O

When interfacing a peripheral to a microprocessor the real problem for the microprocessor is not knowing when the peripheral is READY.

That is, the peripheral operates asynchronously with respect to the microprocessor.

One solution is to program the CPU to repeatedly poll the peripheral’s BUSY/READY flag.

However, this has a built-in disadvantage in that all the resources of the processor are devoted to waiting for this flag. No other tasks can be performed.

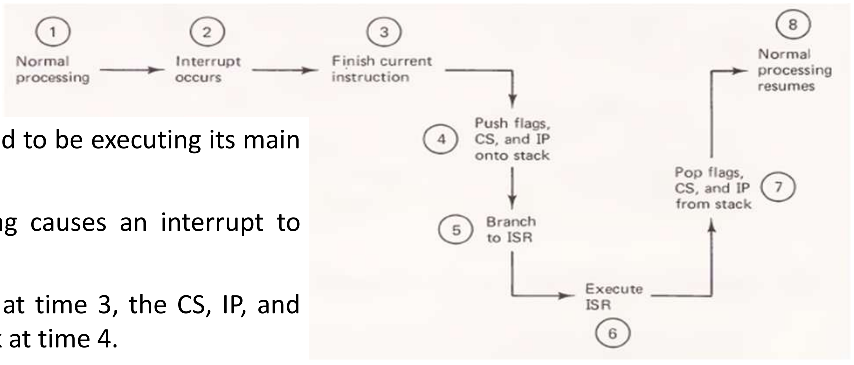
If the peripheral is READY only once every 10,000 ms, the CPU will spend most of its time waiting.

A more logical approach would be to have the peripheral “tell the CPU” when it is READY.

This is the purpose of the microprocessor’s interrupt input.

At the end of each instruction the processor examines the line. If the line active, control is transferred to a special ***interrupt service routine*** (ISR).

The following figure shows the CPU's response to an interrupt.


1
Normal processing
2
Interrupt occurs
3
Finish current instruction
4
Push flags, CS, and IP onto stack
5
Branch to ISR
6
Execute ISR
7
Pop flags, CS, and IP from stack
8
Normal processing resumes

During time 1, the processor is assumed to be executing its main task.

At time 2, the peripheral's READY flag causes an interrupt to occur.

After finishing the current instruction at time 3, the CS, IP, and flag registers are pushed onto the stack at time 4.

Control then transfers to the ISR at time 5.

During time 6, the ISR is executed, terminating with the instruction IRET (interrupt return).

The CS, IP, and flag registers are recovered from the stack during time 7.

At time 8, the original task is resumed.

If we assume that 100 ms is required to respond to the interrupt and supply the peripheral with data, then in the case of the 10,000 ms/character printer, 9,900 ms will be available to the processor for its main task.

In effect, the processor can perform two jobs at the same time.

The 8086 has two interrupt pins, labeled INTR and NMI.

Types of Interrupts

The 8086 has seven different interrupt types, as listed below.

SNO	Interrupt	Initiated by
1	NMI (Non-maskable interrupt)	External hardware
2	INTR	External hardware
3	INT n	Internal via software
4	INT 3	Internal via software
5	INTO	Internal via software
6	Divide-by-0	Internal via CPU
7	Single-step	Internal via CPU

Response Time

The interrupt response time or interrupt latency consists of the time to perform the following:

1. Finish executing the current instruction.
2. Perform two interrupt-acknowledge bus cycles.
3. Push the flags, IP, and CS onto the stack.
4. Calculate the vector table address and transfer control to the ISR.

5. Direct Memory Access

To appreciate the DMA concept, you must understand that the real bottleneck in the data transfer process is the microprocessor itself.

When a text file is output to a disk drive, we are concerned with transferring data from memory to that drive.

Yet with the programmed or interrupt-driven I/O approaches, that data must first be read from memory into the CPU and then transferred to the disk drive.

The microprocessor is an unnecessary “middleman” in this process, with the result that the transfer rate is decreased.

In the direct memory access (DMA), the peripheral is synchronized to main memory, not the microprocessor.

The DMA approach is to “turn off” the processor and let the disk drive directly access the data file in memory itself.

If the memory can supply a new byte of data every 200 ns, data can potentially be transferred at a rate of 5 million bytes per second, directly to the disk drive!

The 8086 has protocols for transferring control of its buses to the DMA controller (DMAC).

Once the DMAC takes over the buses, there are several different types of DMA operations that can be performed.

DMA Protocols

1. Minimum mode protocol
2. Maximum mode protocol

DMA Protocols

1. Minimum mode protocol

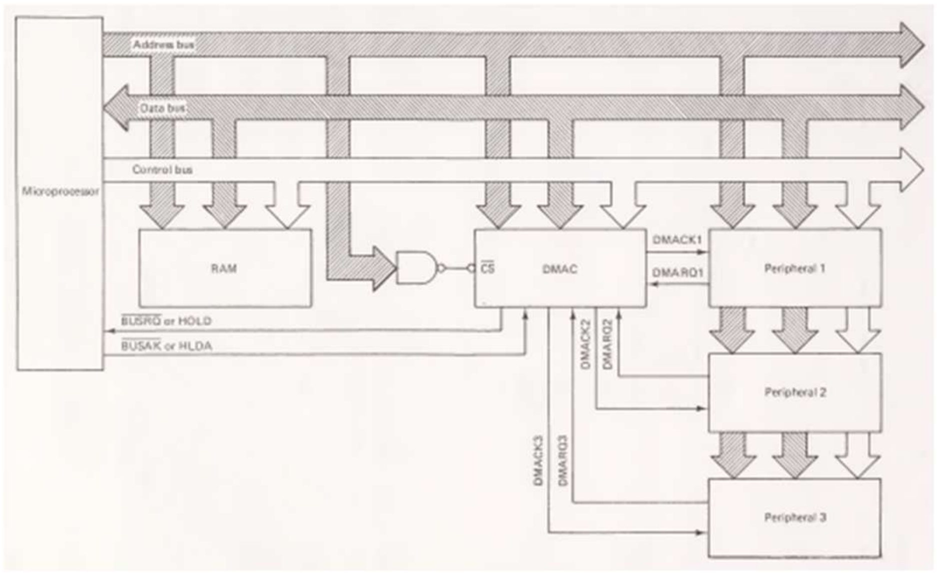
The term protocol represents the handshaking mechanism used by the processor to release control of its buses to another processor.

In the minimum mode two control signals are available: HOLD and HLDA.

DMA Protocols

1. Minimum mode protocol (Contd …)

The following figure shows how the HOLD and HDLA signals are used by the DMAC.



DMA Protocols

1. Minimum mode protocol (Contd …)

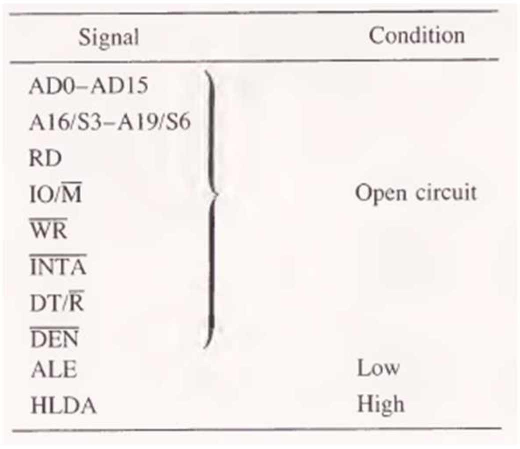
The protocol cycle begins with the peripheral requesting service via the DMARQn (DMA request) input of the DMAC.

The DMAC, in turn, drives the 8086's HOLD input high, requesting that the CPU enter a HOLD state.

The processor responds by finishing the current bus cycle (if any) and then open circuits (tri-states) its address, data, and (most) the control signals as shown in the table below.

HLDA is then output high by the 8086, acknowledging the hold request.

In a system with address, data, and control bus buffers, HLDA is used to disable these buffers so that the CPU is completely disconnected from the memory and I/O.



Signal	Condition
AD0–AD15	Open circuit
A16/S3–A19/S6	
RD	
$IO/\overline{M}$	
$\overline{WR}$	
$\overline{INTA}$	
$DT/\overline{R}$	
$\overline{DEN}$	
ALE	Low
HLDA	High

DMA Protocols

1. Minimum mode protocol (Contd …)

Upon receiving HLDA, the DMAC applies DMACK (DMA acknowledge) to the peripheral requesting service. This is done via the chip select input of the peripheral.

The DMAC is now in control of the system, outputting all the control and address bus signals.

The DMAC is normally programmed by the 8086 prior to the DMA operation for a particular type of transfer.

DMA Protocols

2. Maximum mode protocol

In this protocol, the 8086 supports a request-grant-release protocol using the bidirectional control lines RQ0/GT0 and RQ1/ GT1.

This is a more sophisticated technique that the HOLD-HLDA scheme used in the minimum mode and can be used to allow three processors (including the 8086) to access the local buses.

Upon receipt of a request pulse, the 8086 open circuits its address/data lines, the SO, SI, and S2 status signals, and the RD and LOCK control signals.

Types of DMA

Two types of DMA cycles are possible: sequential and simultaneous DMA.

Sequential DMA	Here, the DMAC first performs a read operation, fetching the data byte into the DMAC. Next, a write operation is performed, transferring the data byte to the I/O port. The opposite sequence is also possible: read a byte from the I/O port, write the byte to memory.
Simultaneous DMA	Here, the fastest transfers are provided. With this technique the read and write operations are performed at the same time to be active simultaneously. In this way data does not flow through the DMAC at all, but directly from memory to the I/O port (or vice versa).

In either case the data transfer is done completely in hardware involving only the DMAC, the peripheral, and main memory.

Because the CPU is not involved, there is no software overhead.

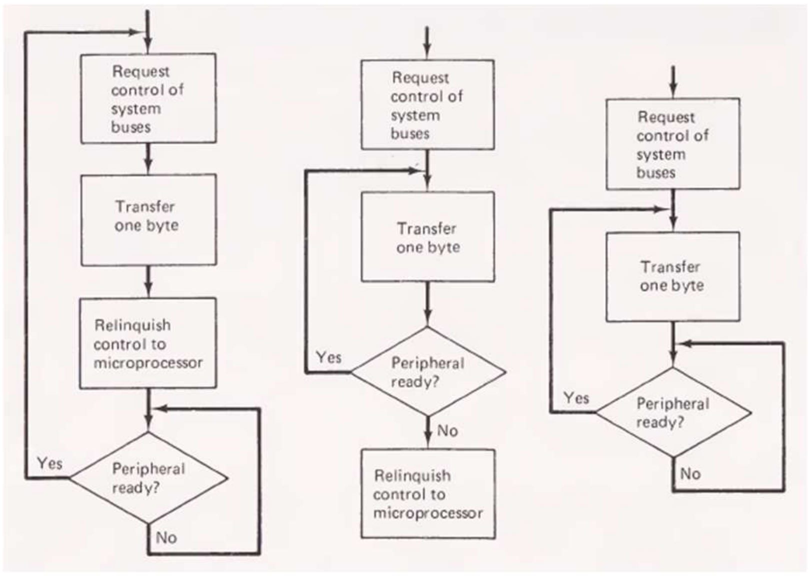
DMA transfer combinations

Several DMA transfer combinations are possible:

1. Memory to peripheral
2. Peripheral to memory
3. Memory to memory
4. Peripheral to peripheral

Before the data transfer can occur, the CPU must program the DMAC for the type of transfer that is to take place, the destination and source addresses, and the number of bytes to be transferred.

Three modes of DMA operation


Request control of system buses
Transfer one byte
Relinquish control to microprocessor
Peripheral ready?
Yes
No
Request control of system buses
Transfer one byte
Peripheral ready?
Yes
No
Relinquish control to microprocessor
Request control of system buses
Transfer one byte
Peripheral ready?
Yes
No

1. Byte mode 2. Burst mode 3. Block mode