MICROPROCESSORS: FROM BASIC CHIPS TO COMPLETE SYSTEMS

R. W. Dobinson,*) University of Illinois, Urbana, Illinois, USA.

"Good-morning, good morning!", the General said
When we met him last week on our way to the line.
Now the soldiers he smiled at are most of them dead,
And we're cursing his staff for incompetent swine.
"He's a cheery old card," grunted Harry to Jack
As they slogged up to Arras with rifle and pack.

* * * *

But he did for them both by his plan of attack.

Siegfried Sassoon
April 1917

1. AIMS OF THESE LECTURES

Microprocessor technology has, since its conception and birth in the early 1970's,
entered very many areas of our lives. No end to its growth is in sight, and new uses
appear almost daily. The semiconductor industry continues to produce ever more powerful
integrated circuits (known far and wide as chips); more functionality and speed at lower
cost is every salesman's cry.

These lectures aim to present and explain in general terms some of the
characteristics of microprocessor chips and associated components. They will show how
systems are synthesized from the basic integrated circuit building blocks which are
currently available; processor, memory, input-output (I/O) devices, etc.

It is not my intention to discuss in detail the many different microprocessors now
available on the market, nor will a complete catalogue of support chips be presented.
Time will not permit this. Instead, emphasis will be placed on explaining the basic
principles of different types of chip. As far as possible I will avoid talking too much
about any specific devices; thus I will spend some time discussing a generic
microprocessor accessing generic memory and talking to the outside world via generic I/O
devices. Some examples of real chips will, however, be given. You may find a slight bias
towards INTEL products in these lectures — this is rather natural; the company was the
pioneer in microprocessor developments and continues, with Motorola, to be an industry
leader both in terms of performance and proliferation of parts.

*) Permanent address: CERN, Geneva, Switzerland.
2. SOME NECESSARY HISTORY

In 1947 the work of Bardeen, Brattain, and Shockley culminated in the demonstration of a solid-state amplifying device—the transistor. Compared with the vacuum tube the transistor had greatly reduced physical size and power requirements. This discovery, one of the most significant of the century, was the first step towards dramatically reducing the size of computers, from an air-conditioned room to a tiny piece of silicon.

Around 10 years after the discovery of the transistor, the first model of an integrated circuit was seen. Integrated circuits are devices containing many transistors on a single chip of semiconductor material. They were first produced in quantity in the early 1960's.

The integration of more and more circuitry on a chip rapidly advanced, in particular under the impetus of NASA and military programs in the US, where miniaturization was vital and cost a non-essential factor. Small-scale integration (SSI) with a complete gate on a chip arrived in 1964. The term SSI implies up to 10 transistors per chip. Medium-scale integration (MSI), a complete register on a chip, appeared on the scene just a few years later; MSI is normally taken to mean 10 to several hundred transistors per chip.

The technology for fabricating large-scale integration (LSI) products commercially existed by the turn of the 60's decade: several hundred to around ten thousand transistors per chip. Early products included a 1 kbit memory, a chip for serial communications (a so-called UART), and chips for calculators. However, the semiconductor industry rapidly became aware of a serious design problem. The complexity of random logic designs in LSI was high and increasing steadily. The development costs, the time, and the skills required to implement new chips were considerable. Unless there was a large market for a chip, manufacturers could not recover their high initial costs. On the other hand, highly complicated LSI chips dedicated to performing one single function would seldom result in high-volume production. Computer-aided design was seen as a possible solution; chips could be designed more quickly and cheaply, an idea we are again seeing emphasised strongly today. Another possible solution was the use of discretionary wiring or master slice approaches, in which a basic circuit module is combined with a final custom layer of interconnects to produce a variety of designs. In fact the industry's dilemma at that time was resolved, as we shall see, by moving in the direction of programmable devices. Enter the microprocessor chip.

The genesis of the microprocessor was sparked off by a contract between Busicom, a now defunct Japanese calculator firm, and INTEL, then a young semiconductor company specializing in memory products. Busicom wanted INTEL to produce a chip set of 12 ICs for a family of high-performance programmable calculators. Typical calculator chips in production at that time contained around 600-1000 transistors per chip. INTEL thought they could double this and were led to propose an alternative solution to the 12-IC-chip set proposed by Busicom. They suggested and implemented a three-chip approach; a flexible programmable (micro) processor chip and two memory chips, one for program and one data. Later a fourth chip was added for output expansion. The microprocessor chip was designated the 4004, and in its final form it contained about 2300 transistors. INTEL, having renegotiated their original contract with Busicom, proceeded to sell the MCS-4 chip set to other customers for non-calculator applications, and the microprocessor boom duly
commenced. INTEL had some misgivings about the viability of making the chips generally available, apparently on the grounds that it might not be economical or possible to compete with then existing minicomputers. The 'genius' behind the decision to go ahead was based on a realization that the microprocessor potential was not as a minicomputer replacement but as a way of bringing intelligence to many new areas.

The INTEL 4004 chip is thus the grandfather of all microprocessors. It was first advertised to the world in November 1970 in an issue of Electronics News that also announced another significant event: the invention of an 'erasable and re-programmable Read Only Memory'. The 4004 was a 4-bit microprocessor. It manipulated data 4 bits at a time. There were 46 instructions (typical execution time 10 μs) and up to 4 k separate memory locations that could be directly addressed. Within a rather short time some of the shortcomings of the 4004 (for example it had no interrupt capability) were improved on by the INTEL 4040. This and other 4-bit microprocessors, notably from Rockwell and Texas Instruments, entered a large number of applications areas. However, the introduction of new designs in the 4-bit field would seem to be unlikely, except perhaps for high-volume specialized applications; 8- and 16-bit microprocessors are now the norm. Rather soon after the advent of the first 4-bit chips, more powerful 8-bit microprocessors became available at only slightly higher prices.

While INTEL was working on the MCS-4 chip set a parallel development project was under way within the company, which led to the introduction of the first 8-bit microprocessor, the 8008. In 1969 Computer Terminal Corporation (now called Datapoint) contracted INTEL to produce a monolithic processor (a processor on a single chip) for use in a CRT terminal. INTEL's chip was late in arriving and too slow by a factor of 10 for Datapoint's needs. It was thus an apparent failure in meeting its original functional requirements. However, INTEL decided to market the product, supposedly in order to encourage sales of their memory chips, and the chip, the INTEL 8008, was offered for sale in 1972.

The 8008 had a typical instruction time of 12 to 20 μs, no faster than the 4004, but it could address 16 k of memory, 8-bits at a time. The success of the 8008 motivated INTEL and other companies to produce more powerful successors. In 1974 INTEL introduced the 8080, which was a full factor of 10 faster than the 8008 while being backwards-compatible with it. Within a year or so of the appearance of the 8080 several other 8-bit processors of similar or better performance were produced by other manufacturers. Most notable was the Motorola 6800, which was introduced as a direct competitor to the 8080, and the Zilog Z80, which incorporated the 8080 instruction set.

Both INTEL and Motorola have gone on to produce large families of chips based on their original processors. Performances have improved; speed, width of data paths, number of address bits, and sophistication of instruction sets. Today's advanced processors use VLSI technology; 10,000 to 100,000 transistors per chip. Figure 1 shows an INTEL 8086 chip. 8-bit processors have evolved into 16- and 32-bit devices. However, manufacturers have attempted to ensure some degree of compatibility, hardware and/or software, in going from one generation of products to the next. Most microprocessors and support chips are manufactured in the familiar dual-in-line packages shown in Fig. 2.
Fig. 1 The INTEL 8086

Fig. 2 Dual in-line package (DIP) for an integrated circuit
A crucial area for any user of microprocessor chips has been the level of support available from the manufacturer and other third parties. Support can be defined as complementary hardware and software products for the basic processor chip. Hardware support takes the form of compatible memory, I/O and other chips to facilitate the building up of the total hardware system. Software support provides an environment for developing, testing, and running code for use in the system; it covers such areas as language translators and interpreters, run-time monitors and operating systems, debuggers and other utilities. The success and popularity of a particular chip is strongly correlated with the level of support available for it.

This ends our short history lesson. We have seen a little of how and why microprocessor technology came about. In fact the emergence of the technology cannot be said to be primarily based on astute design or advanced planning. The first products were in many ways accidents and even rejects!

3. MICROPROCESSOR BASICS

In this section we review the basic workings of microprocessor chips. We look a little into their internal structure and how they appear to the outside world at their input and output pins. The approach I will adopt is both simplified and rather general. I will not, for example, deal in detail with different types of instruction set or addressing modes. However, I hope that the following will enable you to appreciate the properties and capabilities of a generic microprocessor chip. We shall see more of the coupling between the microprocessor and support chips in later sections.

What does a computer do? What are its basic components? It takes in information from the outside world, through input devices; performs some arithmetic and logic operations on the information, using memory and the central processor unit (CPU); and returns results to the outside world via output devices. Figure 3 illustrates these very basic ideas. The CPU is the heart of the system. It is responsible not only for arithmetic and logic operations using its arithmetic and logic unit (ALU) and internal memory registers, but also for the control and timing of all actions throughout the system. Memory is composed of blocks of storage locations which contain the program, a sequence of instructions to the CPU which it normally executes one at a time, and data which is processed by the CPU. Instructions consist of two parts: an operation code or opcode, which specifies what action is to be taken; and operands, units of information operated on by the CPU when it executes an instruction. The instructions can be classified into the following categories (some examples are given in each category).

a) Arithmetic instructions: add, subtract, multiply, divide, increment, decrement, negate, complement.

b) Logical instructions: AND, OR, exclusive OR, bit manipulation and testing.

c) Information transfers: information is moved from one location to another without being changed. Sources and destinations for the information can be CPU internal registers, memory, and I/O devices.

d) Branches in normal sequential program flow: unconditional jumps, conditional jumps, subroutine calls, subroutine returns, software interrupts.

e) CPU control: no operation, halt, enable/disable interrupts.
The number of operands specified for any opcode can vary from none to several. Normally 'several' in microprocessors means up to two: a source and a destination for a move; two sources of data for an addition with the results going to overwrite one of the original sources. Operands are specified in terms of addresses of memory locations, internal CPU registers, and I/O ports. The final effective addresses to be used in accessing information can be specified directly as part of the instruction or indirectly via pointers. For example, an address specified in an instruction may point to another address which contains information (data) to be operated on. A still more complex way of specifying an effective address for an operand is indexing; the effective address is the sum of a special register in the CPU and an offset specified as part of the instruction. We see in general that the effective address of an operand is the result of some address computation. Later we will come back to how instructions are fetched, decoded, and executed. Let us now turn to a more detailed picture of how a computer system is put together, in particular the internals of a microprocessor chip.

Figures 4a and 4b show a typical microprocessor-based system. Part (a) of the figure represents the functions normally incorporated in a 'standard' microprocessor chip, the CPU. The registers shown are representative rather than definitive. Part (b) shows various functions that can be added to complete the system. They are in a 'standard system' provided as separate chips. This having been said, we remark that since 1976 it has been possible to incorporate some of these functions on the processor chip itself. One has traditionally referred to the resulting device as a single chip microcomputer. A
Let us now go in turn through the different components of a standard microprocessor chip. A single internal data bus is shown as providing a communications path between the different registers and the ALU. Operands are brought from registers and from memory to the ALU, and afterwards the results are returned to registers or memory via the same bus. Often one of the registers, the accumulator(s), plays a special role — it provides one of the inputs to the ALU and results are returned to it. Note that the use of a single bus requires the least space on a chip. As space is a critical factor when designing and producing a chip, most microprocessors have a single-bus architecture. The disadvantage of this is slower operation. The bus is multiplexed; it is used for different purposes at different times. Improved execution speed would result if dual or...
even triple buses were used (see Fig. 6). In the case of a dual bus system, a source bus connects to both inputs of the ALU and a separate destination bus is provided. A triple-bus system provides separate source buses for each input of the ALU and an extra destination bus.

In addition to the number of internal buses, their width is an important factor when discussing execution speed. Early chips manipulated 4-bits of information at a time. Subsequent, ever-higher performance chips have used 8-, 16- and finally 32-bit wide

---

Fig. 5 The INTEL 186 chip. A powerful processor and support functions on a single chip.

Fig. 6 a) Single-bus architecture; b) Double-bus architecture; c) Triple-bus architecture.
internal data registers and buses. Execution speed also depends on the type of semiconductor technology used.

The internal data bus is connected to the pins of the microprocessor chip to form an external (or component level) data bus. The internal data-bus width is normally the same size as its external brother. However, this is not always so, and chips exist where, for example, information is manipulated internally 16 bits at a time, yet externally memory access and I/O access take place using an 8-bit bus. One internal bus operation then maps onto two external bus cycles.

Also shown in Fig. 4 are so-called address registers or pointers. They are used to form the external address bus of the chip. The contents of these registers is loaded via the internal data bus. Address registers can be used in various ways to point to the operands of an instruction. We have seen how they can be used in calculating the effective address of an operand. There are two special address registers which I want to mention explicitly: the program counter and the stack pointer.

The program counter keeps track of the instruction stream. Its presence is indispensable and fundamental to program execution. We will see a little more of how this register is used later. A stack is a last in/first out (LIFO) structure. It accumulates data in the order that it is deposited. The oldest data are located at the bottom of the stack; the most recent at the top (see Fig. 7). A stack works in a similar way to a plate dispenser in a restaurant, where plates are piled up in a circular hole equipped with a spring at the bottom. A stack is manipulated by two operations: a PUSH deposits something onto the stack; a POP removes the top element from the stack. The most usual way to implement a stack is as a structure in (external) memory. The base of the stack is set up by the program when execution starts by loading an address into the stack pointer (SP). Each time a PUSH operation is performed, data are written into the address pointed to by SP; the pointer is then decremented — decremented because most stacks grow down from a high memory address base. Each time a POP is done SP is incremented and the corresponding data item read from the memory location. Stacks are used to pass subroutine arguments and as part of the so-called interrupt facilities present in all microprocessor chips. We shall discuss interrupts in the next section.

![Implementation of a stack](image)
The instruction register holds the instruction being currently executed. The control unit decodes the instruction and carries out the actions which are required. It is responsible for generating the necessary synchronization signals and manages all transfers between I/O, memory, registers, and ALU. Synchronization occurs with respect to a clock; this is a precise periodic signal of around 2 to 16 MHz, derived from a quartz crystal. There are control signals from the control unit to all parts of a microprocessor chip. These internal control lines have not been explicitly shown in Fig. 4a to avoid cluttering up the diagram. However, what is shown is a control bus going to the outside world, where, together with address and data buses it forms the component level bus. We will be considering the properties of this bus in more detail in a short while. Two main techniques are used to design the control unit: handwiring and microprogramming. Most microprocessors use the latter technique. The sequencing function of the control unit is implemented by a special internal program called a microprogram; this is stored in an internal Read Only Memory (ROM) which you can see in Fig. 4a. The instruction register contents are used in the ROM to point to an appropriate sequence of lower level microinstructions. These microinstructions act at the gate, register, and bus level to carry out the required instruction. The usefulness of microcoding the instruction set of a processor can be summarized as follows:

1) It improves utilization of the limited chip area by providing more logic per square millimetre than a random logic design. This is because it is easier to build regular structures such as memories in a compact way.

2) The microcode burned into the ROM can be changed relatively easily by the manufacturer. This is a major advantage in correcting early design mistakes or for adding instructions and/or improvements at a later date. It is even possible to emulate an entirely different instruction set for the microprocessor chip. For example, a Motorola 68000 has been microprogrammed to emulate part of the IBM series 370 instruction set. Some minicomputers allow end users access to microcode on their machines. If you have a spare year or so you can amuse yourself adding your own instructions. At the microprocessor chip level the end user does not get to touch the internal microcode—not unless your name is IBM.

There is one more register I want to mention: the processor status register which appears in Fig. 4a in association with the ALU. This register contains various flags which define conditions that have occurred in the ALU as a result of an instruction being executed; for example: carry, overflow, negative sign, zero result. Other control bits may exist and normally include a bit for enabling and disabling processor interrupts. The processor status register is often called the processor status word or PSW. It is accessible from the internal data bus, and in fact can be tested and manipulated implicitly or explicitly by certain instructions, e.g. branch if zero result from last arithmetic operation, enable interrupts, etc.

We are now in a position to see how an instruction is executed by our microprocessor chip. Be warned that what I am going to say is very much simplified. In addition, different processor chips work in different, sometimes more complicated, ways. What I will present does however follow fairly closely what happens in an INTEL 8080.
The execution of every instruction starts with an instruction fetch phase. The starting address of the next instruction to be executed is contained in the program counter whose contents are gated out onto the external address bus together with appropriate control signals during an external memory access cycle. The contents of that memory location are placed in the instruction register (IR) and the program counter is incremented to point to the next location. The opcode contained in the IR is used to trigger a sequence of corresponding micro-operations. Each step in the sequence corresponds to an internal state. Each state is associated with the execution of a microinstruction and we assume lasts one tick of the master clock. Depending on the instruction being executed, further external memory cycles may be required to fetch source operands associated with that instruction. These operands may be contained in successive memory locations, in which case they are pointed to by the program counter, or they may be accessed via other address registers. The required operation can now be performed using the ALU. If an external memory location has been specified as a destination operand in the instruction, a memory cycle is required to store the results of the operation.

Some comments are in order about the instruction fetch and execution sequence just described. If the instruction specifies information transfer — a move between internal registers, memory, and I/O — then strictly speaking the use of the ALU is not necessary. If the source and destination operands are contained in internal registers then no further external memory cycles are required after the instruction fetch. Instructions which specify branches in normal program flow will change the value of the program counter explicitly to a new, non-sequential, value. In the case of branches to subroutines, the return address — the current value of the program counter after fetching the branch and its operands — is normally stored automatically on the stack. A return from subroutine instruction automatically restores the program counter from the stack.

In this section we have looked at the internal workings of a standard microprocessor chip. We have identified its component parts, the internal buses which interconnect these component parts, and described how the external component level bus of the chip is derived. We also saw in a very simplified way how instructions were executed. No further details will be given of chip internals. We will now go on to focus attention on how a chip appears at its pins and how it is coupled through these pins to support devices.

4. MICROPROCESSOR BUSES

We have spent some time looking at the internal components of a microprocessor chip. It is now time to consider how a chip talks to the outside world and how the outside world talks back. We introduced the concept of address, data, and control signals emerging from the microprocessor chip as shown in Fig. 8. This component level bus is now discussed in more detail.

The first obvious function of the bus is to transfer information into and out of the chip, to and from memory and I/O devices. In a basic transaction the processor asserts an address, and data are either written to or read from that address. Control lines are used to command the required direction of data transfer and to synchronize or time the whole operation. Figure 9 summarizes these basic ideas.
The address information provided by present-day processors typically comprises between 16 and 32 parallel output lines from the chip. These lines provide a unique identification for between $65536$ and $4295 \times 10^6$ different locations. Some microprocessors make a distinction between how memory and I/O devices are addressed. They do this by providing a second I/O address space. This space, which is normally considerably smaller than the main address space, could be selected for example via extra control bus lines. Note that Fig. 9 shows an I/O address space command line. Special I/O instructions are available for accessing devices located in I/O space. Processors which do not provide a separate I/O space assume that you will allocate both memory and I/O device addresses from the single main address space. You can, if you so choose, ignore I/O space even when the chip provides it for you. Treating memory and I/O device locations in the same way is sometimes an advantage. The I/O operations are then referred to as memory-mapped. However, there can also be advantages in keeping the two types of operation separate; when there are only 16 bits of main memory address information, a user may be reluctant to sacrifice part of the space unnecessary for I/O — in some cases DMA transfers between memory and I/O are more convenient if distinct I/O addressing is used.
The data-bus width is slightly more variable in size than that of the address bus. It ranges from one byte wide on the low end and some medium performance chips, through 16 bits for the INTEL 8080, 186, 286, and earlier Motorola 68000 series, to 32 bits for the latest Motorola 68000 offering, the 68020.

It may be surprising to learn that one of the main constraints imposed on the design of LSI and VLSI components was, for a long time, a limitation of around 40 pins per chip for input and output signals. This was mostly due to the fact that industrial testers would not accept packages with more than 40 pins. You can see that even with a modest 16 bits for data and 16 bits for address you will run into problems, given the rather complex functions required from the control bus. The control bus is rather more than just timing and a Read/Write line, as we shall shortly see. Additional pins are also required for power, ground, and the chip master clock. This has led in some cases to the use of pins being shared between two different functions: part of the time for one purpose and part of the time for some other purpose. The most commonly met scheme is to multiplex address and data, although the use of other pins may also be shared. The impact for a designer of sharing of pins between address and data is that he or she must 'catch and hold' the address, which comes out of the chip first and remains only fleetingly. This is done by latching the address with the help of a special timing line provided from the chip as shown in Fig. 10. This procedure effectively demultiplexes the address and data and

---

**Fig. 10** Use of the ALE line on an INTEL 8086/88 to demultiplex address and data. The address, which appears briefly at the start of a transaction, can be latched on the low-going edge of ALE. Note that the high address lines are used to provide status information later in the transaction, and the low sixteen address lines are used for data transfers.
allows the designer to work with separate address and data lines thereafter. Even with the removal of the 40-pin restriction there still seem to be insufficient pins to go round, and many current (and even new) microprocessor and support chips use multiplexing for getting signals into and out of a chip. We note one final point in passing: dual in-line packaging is rapidly being replaced by other 'alternatives' when large numbers of pins are involved.

There are fairly substantial differences between microprocessors as to how data transfers into and out of a chip are synchronized. Figure 11 compares the timing of a 'Motorola 6800 type' bus with that of an 'INTEL 8080 type' bus. The processor chip acting as a master reads or writes data at an external slave device. For the 6800-type there is a Read/Write command line specifying the direction in which the data transfer should occur, and a separate strobe line to specify when things should happen. Data can be accepted by an external slave, memory, or input/output device, on the falling edge of the strobe for a write. Valid data must be available from the slave by the time the rising edge of the strobe occurs for a read cycle. The 8080-type bus uses separate read and write signals which specify what is to be done and when. They are thus timing/command lines used to set up the direction of the transfer and to trigger it. In addition there are actually two pairs of Read/Write lines, one for addressing the main memory address space and one for addressing I/O space. In both examples that were discussed, the timing signals are derived from the microprocessor internal clock. Both schemes shown represent synchronous timing. There is an implicit fixed timing agreement between the processor (master) and external slaves. The processor completely defines the bus cycle time and assumes that a slave can accept or provide information when required to do so. This is a simple and fast method of timing but assumes that everyone can keep up. The disadvantage

---

![Diagram of bus timing](image)

**Fig. 11** Comparison between the timing of a) an 'INTEL 8080 type' bus with b) a 'Motorola 6800 type' bus.
is that if slow slaves are to be accommodated then the processor must be slowed down to a speed acceptable to the slowest slave. This reduces the speed of all transactions — particularly annoying if a slow device is only rarely addressed.

Semisynchronous timing allows a slave, who cannot respond in time to meet the normal timing requirements of the processor, to hold up, modify, this timing in some way until it can respond. The recognized procedure is for such a slave, when it is addressed, to assert a 'wait' line. The microprocessor cycle is then held up (stretched) until the wait line is negated, whereupon it continues and terminates its cycle as usual. Figure 12 shows the bus timing of the Motorola 6809. Notice how MRDY is pulled low by the slave and held there until it is prepared for the cycle to continue. At this time MRDY is allowed to go high. Signals E and Q, which are essentially the microprocessor clock, are thus stretched by the action of MRDY. It is interesting to note that there was no wait line on the original 6800 chip, a significant omission. Almost all the current microprocessors implement some form of semisynchronous bus cycle, although the precise mechanisms will differ from that described for the 6809.

One other type of timing is sometimes used: asynchronous timing which involves a handshake between the processor (master) and a slave device. The processor informs the slave when something is to happen by asserting its timing signal, and the slave asserts its timing signal when it is in order for the processor to continue with the cycle. The well-known example of this type of timing is the Motorola 68000 family. This is illustrated in Fig. 13. The processor clock is shown at the top of the figure. The next two signals shown are the address lines, and the address strobe which indicates that the address is valid. The upper and lower data strobes (UDS and LDS) indicate that data are

![Figure 12](image-url)

**Fig. 12** Timing for the Motorola 6809: a) shows how Reads and Writes are normally carried out; b) shows how a slow slave can stretch bus cycles.
being transferred on the upper, the lower, or both bytes of a 16-bit word, and the Read/Write line gives the direction of transfer. The data acknowledgment (DACK) input from a slave to the processor is a handshake signal. Until it is asserted — pulled low — the 68000 will insert so-called wait states. You can see this clearly in the slow read cycle shown in Fig. 13. Also shown in the figure are 16 data lines and the FC function code lines which identify the type of bus cycle being performed. One of the interesting points about the 68000 is that along with its basic asynchronous mode it supports an alternative synchronous timing mode compatible with the 6800 family. This allows 6800 support chips to be used with the 68000.

Let us now summarize the three types of timing we have discussed.

<table>
<thead>
<tr>
<th>Type</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td>Here is the address; give me the data before time T.</td>
<td>Here is the address and data and I hope you catch it.</td>
</tr>
<tr>
<td>Semisynchronous</td>
<td>Here is the address. If you can not give me the data before time T tell me by t.</td>
<td>Here is the address and data. I hope you catch it. If for any reason you need more time tell me by t.</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>Here is the address. I will hang around until you tell me the data is valid.</td>
<td>Here is the address and data. I will hang around until you tell me you got it.</td>
</tr>
</tbody>
</table>

Fig. 13  Asynchronous timing of Motorola 68000
Our initial discussion has focused on a situation where the microprocessor chip acts as a master the whole time, continuously able to generate component level bus signals to the outside world. There are, however, situations where the chip may be sharing access to external devices. For example, there may be another processor or master wanting to perform a read or write to memory or I/O devices. This often occurs if the CPU is used together with a DMA controller, an I/O processor, or a floating-point processor. Figure 14 illustrates this point. The control bus of most processors therefore provides some sort of arbitration bus signals to ensure that contention between competing masters can be resolved in an orderly and sensible way. A very common method is to use two lines: a hold request (HOLD) signal going into the microprocessor chip, and a hold acknowledgement (HOLDA) coming back. HOLD is asserted by a device external to the microprocessor to request use of its component level bus. When the microprocessor has finished its current activity it will effectively disconnect from its component level bus pins and assert HOLDA. The external device then uses the shared bus and, when it has finished, negates HOLD. The processor negates HOLDA and may resume using the component level bus. Figure 15 shows the principles behind the use of these two arbitration lines. Sometimes there are several pairs of HOLD/HOLDA lines.

Almost every microprocessor supports some form of interrupt capability. We review the meaning and importance of interrupts and mention briefly some schemes that are in use. Historically, interrupts arose from the realization that input and output operations involving the CPU, the memory, and peripherals could easily waste large amounts of CPU time. Time used in polling loops for I/O devices to accept or make available information, was time lost for computation. The sharing of the CPU between these two tasks could be much more efficiently carried out using a program interrupt facility. This allows the CPU to respond automatically to certain conditions, either internal or external (coming from peripherals). These conditions normally occur at unknown times (i.e. asynchronously). They force the CPU to execute some pre-specified procedure or subroutine. Basically an interrupt can be considered a jump to a subroutine executed by hardware. The interrupt occurs after the current instruction has been executed by the processor, and afterwards program flow continues where it left off. Figure 16 illustrates the basic ideas behind the interrupt mechanism. Usually the program counter and processor status words are automatically saved on the system stack when an interrupt occurs and restored when a return is made from the interrupt service routine. Sometimes a processor will in addition automatically save and restore other internal registers. This can make for faster interrupt handling.
Fig. 15 Use of the arbitration lines HOLD/HOLDA to share the microprocessor bus between the main processor and a DMA controller.

Fig. 16 Basic principles of interrupt handling
When an external device needs to interrupt a microprocessor chip it asserts the processor's interrupt pin(s). Each processor manufacturer has different ideas and implementations for processor interruption so, in detail, different chips work in different ways. When an interrupt occurs, how does the processor know where the interrupt service routine starts? The simplest way is for a processor to branch to a set location when servicing an interrupt. This location should contain the first instruction of the interrupt service routine. An alternative which allows a little more flexibility is to use the fixed location to point to the address of the first instruction of the service routine. Often a single interrupt service routine is inadequate. Newer processors have several locations reserved for this purpose. How is one chosen over the other? One way is to have several lines built into the processor. Each interrupt line has an associated interrupt service routine and a memory location reserved for the routine's starting address. Less lines are used if the interrupt number is coded and this is sometimes done. Another approach is to have the processor run a special interrupt acknowledge cycle. External hardware, either in the I/O devices themselves or in special interrupt controller chips, then provides additional information to help the processor pick one of several service routines. The additional information is typically a number, a so-called vector, which points to one of several alternative entries in an interrupt table maintained in a set of fixed memory locations. The vector could also be the direct address of an entry in the table.

Figure 17 shows the use of an interrupt controller chip in association with a microprocessor. The interrupt controller is used to handle up to eight separate interrupt lines. The controller may be programmed internally via the data bus, chip select, and address line Ao, and the Read and Write strobes RD and WR. The INT interrupt line goes to the microprocessor chip, and in response to the INTA (interrupt acknowledge) a vector is placed on the data bus.
lines from I/O devices. It can be programmed from the processor to enable or disable any combination of the eight interrupt request lines by means of a mask register. The interrupt controller decides which of several simultaneous device requests is serviced first according to several programmable schemes; for example, fixed priority or rotating (round robin) priority. There is an additional priority feature which inhibits any interrupt with lower priority than specified in the 'in service register'. The controller chip will pass on an interrupt request to the 8088 and will respond with the vector corresponding to the device request being dealt with when asked to do so by the processor. A vector can be programmed for each of the eight device interrupts. These vectors point to corresponding entries in the interrupt service table.

We have seen that the control bus has a number of complex functions. First it provides timing and commands for normal bus transactions; secondly, it provides some form of arbitration mechanism; and thirdly, it takes care of interrupts. There are, however, still a number of other miscellaneous lines and corresponding functions which are lumped together under the title of control bus: for example, a reset line to force the processor into some start-up procedure, a halt line to force the processor into an idle state, and status lines from the processor, which inform external devices what operations it is performing at that time (instruction fetch, memory, or I/O access, interrupt acknowledge cycle, halted, etc.). Figure 18 summarizes the appearance, at a component bus level, of a generic microprocessor chip.

So far we have focused attention on the logical description of the I/O pins of a microprocessor — its so-called component level bus. We have looked at the use of address, data, timing, and command lines, and considered how bus arbitration and processor interrupts are handled. In putting together a system on a single printed-circuit board it is usual to use essentially the component level bus for interconnecting the processor and other chips. I use the word 'essentially' because there may, perhaps, be a need to demultiplex address and data or boost the drive capability of certain output lines. However, the system bus can be considered to be the component level bus of the microprocessor.

When a system cannot fit on a single printed-circuit board, an interconnection scheme is needed to allow communication between the multiple boards that now make up the

![Fig. 18 A generic microprocessor chip](image-url)
A backplane bus approach is the most commonly used technology for this purpose. This bus is a printed-circuit board equipped with multiple connectors that provides a shared communications path and power for plug in boards (see Fig. 19). A normal set-up would comprise several boards containing processors, memory, and I/O as shown in Fig. 20. The backplane bus can now be referred to as the system bus. A backplane bus may be very closely related to the component level bus present on a processor board. However there are good reasons for not adopting such an approach. We will briefly review them.

Fig. 19 A backplane bus

Fig. 20 Synthesis of a basic computing system by CPU, memory and I/O printed-circuits boards.
When synthesizing a microprocessor-based computing system it is very convenient to adopt a modular approach for both hardware and software. In particular, at a hardware level one would like to be able to purchase printed-circuit boards from the (different) manufacturers who offer the optimum products for tailoring a system to specific needs. It is an advantage to not be tied too closely to a particular processor, memory type, or I/O device, but to be able to evolve in the light of new technologies and end user needs. Defining a standard backplane bus which is processor- and manufacturer-independent and endorsed by a reputable national or international body is a very positive step towards this goal. Many of you in the high-energy physics world will be familiar with the advantages of standardization in a variety of areas including CAMAC.

The second reason for not using microprocessor-component-level bus signals on a backplane is that in most cases they are just not very suitable for direct use in general purpose multiprocessor (non-hierarchical) systems. The facilities they provide for interruption and arbitration in particular are somewhat rudimentary and require enhancement before they can be considered adequate for many applications.

Microprocessor backplane bus standards is an important topic which is being worked on actively by many people, in particular by the Institute of Electrical and Electronic Engineers (IEEE). This body has committees which are defining or improving a number of standards, including STD bus, MULTIBUS, VME bus, S100 bus, and the new P896 bus. Whilst some of these offerings were originally manufacturer and processor inspired, there seems room to be optimistic that the final standards will permit, in the main, a good measure of 'mixing and matching' of products.

5. THE ROLE OF TTL 'GLUE'

We have discussed in some detail the inner workings of microprocessors and their component level buses. In the following sections we will be concerned with the connection of 'naked' microprocessors to a wide variety of other special-purpose VLSI and LSI chips. We shall discuss various types of chips; primary memory, parallel and serial input/output chips, CRT and keyboard controllers, and so on. All these chips, like the microprocessors they support, are made up of transistors — which is the lowest level of abstraction I shall mention. Transistors are interconnected to make a slightly higher level of abstractions, namely gates to carry out the basic logic operations such as NAND, NOR, AND, OR, Exclusive OR (XOR), etc. Gates themselves can be coupled together to form flip-flops, which are binary storage elements. Individual storage elements can be put together to make byte wide registers and counters, and so on upwards to still higher levels of abstraction and complexity, arriving finally at microprocessor and microcomputer chips themselves.

Basic microprocessor systems may consist of just a few complex chips. However, more usually a fair number of simpler chips are used in auxiliary but nevertheless very necessary roles. Let us look at some examples.
1. A microprocessor chip may lack the electrical capability to permit its direct connection to a large array of memory chips. It may be necessary to buffer most lines with line drivers.

2. Additional chips or simple latches may be required to demultiplex address and data lines coming out from the microprocessor.

3. Memory chips and other devices for I/O which are attached to a microprocessor bus must recognize when they are addressed and respond appropriately by gating data to and from the bus. Address decoders, multiplexors, as well as basic logic gates, all find a role here.

The crux of the matter is that microprocessor systems need some 'glue' to stick the larger, more complex chips together. Much of the glue is provided by the well-known 7400 series of TTL digital logic and there are several other compatible lines. You can find the data books of Texas Instruments, Motorola, Signetics, AMD, National, and others, bulging with descriptions of useful chips. Figure 21 gives a minute cross-section of available products. However, this is not meant to be a course in how to design with MSI digital logic so I shall not pursue the topic. Furthermore, precise details, on a pin-by-pin basis, of how microprocessor based systems are put together are not necessary, the aim of these lectures being rather to give an introductory overview to the field. However, do remember the essential role played by the 'glue.'

Fig. 21 Examples of TTL glue. a) Inverter; b) AND gate; c) D-type positive edge triggered flip-flop.
6. PRIMARY MEMORY

Memory implies storage of information, and storage functions exist throughout microprocessor systems. We have seen that internally within a chip there exist data and address registers which are used to store and manipulate information. The role of Read Only Memory in storing the microcode used to implement a processor's instruction set has also been mentioned. Two more general types of external storage can be identified:

i) Primary memory, which is used to store information that a CPU needs 'immediately'. By immediately we mean with an access time of 100 – 1000 ns. Access time is the time interval between information being requested and when it is available. When we talk about memory we normally mean primary memory. We will discuss the different forms of primary memory later in this section. ii) Secondary memory (or secondary or auxiliary storage) is taken to mean devices such as magnetic tapes and disks. Here, access times are much longer than primary memory but the storage density is much larger. We shall discuss one example of secondary storage later.

There exists a storage hierarchy which is depicted in Fig. 22. Notice that two terms are introduced in this figure: random access and sequential access. Random access allows

![Diagram of storage hierarchy](image)
any storage location to be accessed at will without cycling through other locations — there is a fixed access time. Sequential access methods require cycling through all previous storage locations before arriving at the desired information. This type of device has a variable access time. An example of a random access storage device would be the primary memory of a microprocessor system (Read/Write static or dynamic memory). Here, access times are very short. An inexpensive cassette recorder/player is an excellent example of a sequential access storage device. If information is at the start of the tape, the access time is relatively short; if it is at the end, you might have to wait a minute or so. Some storage devices employ both random and sequential access techniques. For example, a disk can move its head to a track in a random way, but reading information within a sector is a sequential operation.

We now look at some very general ideas about primary memory. Almost all memories have a storage matrix as their basic component. The actual storage medium will be discussed later. A matrix arrangement of rows and columns is used to reduce the number of lines required to address (define) a location. A one-dimensional array of 16 memory locations would require an individual selection line per location. However, when a two-dimensional matrix form of storage is implemented, as shown in Fig. 23, only two-row and two-column inputs are required. We say there is coincidence selection. Row and column decoders are used to translate input lines into row and column lines. Storage locations are chosen by a coincidence of a particular row and a column line. Data to be written (placed in) or read (retrieved from) at a particular location are routed to and from the storage matrix via the I/O control section, and a Read/Write input controls the direction of data flow. The chip enable (CE) serves as an additional address input line, allowing the Read/Write line to be effective only when it is asserted. We distinguish three types of memory cells; static and dynamic read/write memory, called RAM (random access memory), and Read Only Memory referred to generically as ROM.

Static memories use storage cells that resemble flip-flop storage circuits. (Flip-flop binary storage elements were briefly discussed in the previous section.) Such storage cells allow non-destructive information readout. The stored information remains in the cell as long as the memory array is powered on and the location is not explicitly changed. The storage cell is volatile in that when power is turned off the information is lost. Some media such as magnetic cores, which were used up to a decade ago, were non-volatile. Nowadays battery back-up can be used to alleviate problems of information loss in volatile semiconductor memories where this is deemed necessary. Different semiconductor technologies result in memories with different access times and power requirements. Access times vary from ten to a few hundred nanoseconds.

Dynamic memory cells store information using the absence or presence of electrical charge on a capacitor. Unlike static memory the cells cannot retain data indefinitely. This is because the charge tends to leak away with time and it is necessary to refresh the memory periodically in order to preserve data integrity. The refresh problem and its solution are discussed with reference to Fig. 24, which is a simplified representation of what actually happens. The capacitor C is the storage element. To write data, one closes switches S1 and S2. A logic one at the input charges C, a logic zero discharges C. When the switches are open, C is ideally isolated. To read data the output switch S3 is
Fig. 23 General ideas about memory.  a) Tabular form (linear selection); b) Matrix form (coincident selection).

Fig. 24 Symbolic diagram of the structure of dynamic memory.
connected to a comparator which interprets the stored charge as a one or zero. Reading
the cell contents will disturb the stored charge on C. Therefore the Read must be
followed by a recharging of the capacitor via closure of switches S4 and S2. The problem
with dynamic storage is that in practice there is leakage of charge from capacitor C over
a period of time, and eventually the stored data are lost. Note, however, that a Read
operation will refresh the cell. Dynamic memories available today have minimum refresh
rates of one refresh every 1 to 2 ms. Clearly it would be impossible to refresh each cell
individually; this would work out at once every 15 to 60 ns for the 16 k- and 64 k-bit
chips in use today. What is done instead is to refresh all bits in a row of a matrix
array when any bit in the row is read. For a 128 by 128 array, one gets a row refresh
interval of 8 to 16 us. For a memory with a cycle time of 200 + 500 ns the refresh only
occupies a small percentage of the available memory bandwidths. Recent, even larger
memory arrays have kept the refresh load to a similar low level by increasing the minimum
overall refresh interval out to 4 ms or by refreshing two rows at once instead of only
one. Dynamic RAMs require extra circuitry compared with static RAMs in order to take care
of the refresh problem. Although the controller required for this can be supplied in the
form of a special-purpose memory controller chip with integrated refresh facilities, the
use of dynamic memories may be unnecessarily complex for small systems where a few static
chips may suffice. They are more suitable for larger memory systems. The clear advantage
of dynamic memories is that they use less transistors and less power per memory cell than
do static memories. Dynamic RAMs typically give a factor of 4 more bits per chips than do
static RAMs. The maximum capacities for present-day chips are 256 k-bits for dynamics
compared to 64 k-bits for statics. Some dynamic memories have on-chip refresh aids. These
range from circuits which, when externally pulsed via one of the chip's pins, will refresh
each row in turn, to the complete on-chip refresh controllers used in so-called integrated
RAMs (IRAMs). In the latter case the refresh mechanism is provided in a way which is very
largely transparent to the world outside the chip.

Figure 25 gives a more detailed picture of a RAM. We use this diagram to represent
both static and dynamic memories, symbolizing the need for refresh circuitry in the case
of the latter by a dotted box at the top of the figure. So far we have considered that
each address corresponds to a single memory cell, i.e. one bit of information. We can
easily imagine a situation where one address corresponds to more than one bit of
information; calling up a storage location allows access to several binary memory cells.
Notice in Fig. 25 how a row, once selected, supplies four bits of data. Instead of
selecting one of these four bits via the two-column address lines, one could bring out all
four bits. Thus one could end up with a 16-bit chip arranged as four address locations of
4 bits, instead of 16 locations of one bit. Figure 26 shows a more realistic situation
where we have a 4 k-bit chip, and a 64 × 64 array organized as 1024 addressable words each
of four bits. You might like to visualize this as four parallel matrix arrays, each
64 × 16 × 1 bit. Apart from the previously discussed need for periodical refresh, dynamic
memory chips impose a further complication, namely that they employ address
multiplexing. In order to save on the number of input/output pins on a package, and thus
reduce the chip size and augment the density of large memory systems, the address is sent
in two parts on the same lines. First the row address is sent, then the column address.
Each part of the address has an associated control line; one replaces a single chip select
Fig. 25  More detailed picture of the structure of a generic RAM chip

Fig. 26  A 4 K-bit chip. A 64 x 64 array organized as 1024 addressable words of 4 bits.
line by a row select plus a column select input. Figure 27 shows a comparison between a generic 4 k static RAM and a 64 k dynamic RAM chip. Both have separate bit input and output data lines. Also shown is a static 4 k RAM arranged as 1024 x 4 bits. Note, in this latter case, that in order to economize on input/output pins, the four data bits enter and leave the chip via bidirectional pins.

Read Only Memory (ROM) is the third type of primary memory we shall discuss. The original meaning of ROM was a system for storing information in a permanent non-volatile form. The first ROMs contained cell arrays in which the sequence of ones and zeroes was established during manufacture of the chips using a metalization interconnect mask. Users had to supply the ROM manufacturers with an interconnect list; the vendors then built the chips. Set-up charges were high and even prohibitive unless users were ordering on a large scale. To offset this high set-up charge, manufacturers developed user-programmable ROMs called PROMs. The first devices used fusible links that could be electrically melted or burned with a special PROM programmer (see Fig. 28). Like a ROM, a PROM which was faulty or required changes had to be discarded. As an alternative to fusible links, INTEL pioneered an erasable PROM, called an EPROM. An EPROM is programmed to have a one or zero in each cell by storing the corresponding amount of charge. Once programmed the EPROM is expected to retain its charge for 10 years or so. However, in the presence of ultraviolet light which enters the chip through a special window, discharge occurs in 15 to 20 minutes. The EPROM can then be reprogrammed.

The EPROM was obviously not intended for Read/Write applications, but proved very useful for research and development environment where its contents might have to be

---

**Fig. 27** a) 4 K x 1 bit static RAM chip; b) 64 K x 1 bit dynamic RAM; c) 4 K-bit static RAM arranged as 1024 x 4 bits.
altered several times. Another important step was taken when the electrically erasable programmable ROM was introduced. Instead of removing the chip from its socket and exposing it to UV light the PROM could be selectively reprogrammed in situ. The E²PROM therefore acts as a Read/mostly Write/sometimes storage element. The programming, or Write time, is typically ~ 10 ms. The storage will be retained for about 10 years but only a total of \(10^6\) Erase/Write cycles can be performed. The term ROM is often used as a generic term for all types of read only memory.

The electrical connections of ROMs are very similar to those of a static RAM with the exception, of course, that there is no Read/Write pin; the chip is always in Read mode. EPROMs require special pins to permit programming functions.

There are very many applications where the use of ROM is invaluable. Software can be developed in RAM and later, when in a proven, final, form, transferred to ROM or PROM. To facilitate this, there are families of memory chips which have pin-compatible RAMs and (E)(P)ROMs. Small stand-alone applications where microprocessors are acting as a replacement for hardwired logic make particular use of ROMs. Many current home computers store large parts of their operating system and utilities in ROM. Standard network protocols are another item which is increasingly ending up in ROM or PROM. Read Only Memory is available in many forms and sizes up to a massive 256 k-bits per chip.

7. **PARALLEL INPUT/OUTPUT**

In order to connect an input or output device to a microprocessor system it is convenient to provide the following basic facilities:

a) A parallel data path one or several bytes wide.

b) An input latch which keeps data valid long enough for the microprocessor to read them.

c) An output latch to freeze the output data for long enough so that the external device can use them. Remember that a processor has stable data on its bus only very fleetingly.

d) Control and status bits which inform the external device and processor about the availability and flow of data being transferred via the input and output latches. For example, the processor needs to tell a device there are valid data in the output latch; the device needs to inform the processor when it has taken the data and is ready for a new transfer.
e) Various occurrences of control and status bits may need to cause interruption of the microprocessor; for example, the availability of a new byte of data from an external device may be signalled via an interrupt.

We note that the use of latches for input and output of data serves two purposes: it electrically isolates the external device from the processor bus, and by 'buffering' all transfers it removes any speed mismatch between the processor and external devices.

Many manufacturers have provided chips to facilitate parallel I/O. We focus attention on a family of quite sophisticated general-purpose parallel I/O chips which can be programmed (i.e. configured) to carry out a wide range of different roles in microprocessor systems. Such devices are known by a variety of names: PIA (Motorola), PPI (INTEL), PIO (Zilog). We will use the abbreviation PIO, meaning both parallel input/output and programmable I/O. A generic PIO chip is shown in Fig. 29. Its main features can be listed as follows:

a) The chip contains several independent byte-wide data paths called ports. Two or three are usual.

b) Data lines are programmable as to direction. Each line or group of lines can be defined and used as an input or output line by loading appropriate control registers within the chip; for example, a direction register containing a Read/Write bit for every data line.

c) The control and status lines associated with a port may be programmed in an equally flexible way. Various different schemes for handshaking data transfers between a port and an external device may be chosen and implemented. Interruption of the processor may be triggered when a variety of conditions are detected.

In Fig. 29 the upper external connections are to I/O devices; the lower connections are to the microprocessor bus.
Complex as they may seem, PIO chips are conceptually very simple. They are useful in many different applications because of their flexibility, and one sees them as the basis for interfacing a wide variety of devices in microprocessor systems.

8. SERIAL INPUT/OUTPUT

When extremely high data rates are not required the number of wires between an external I/O device and a computer system can be substantially reduced by converting from a parallel data path, one or several bytes wide, to a serial data path where there is one single wire for each direction of data flow. A bidirectional connection thus requires two wires plus a common ground connection. Serial transmission is particularly important over long distances where it becomes impractical to lay multicore cables or where transmission via the telephone system is required. Serial transmission does require extra logic to convert back and forth between parallel and serial data streams. However, owing to standardization of a few widely used techniques and the existence of mass produced (cheap) LSI interface chips the extra cost and inconvenience is negligible. I will briefly review the structure of a generic I/O port and then discuss one implementation which is very widely used today, namely asynchronous character/byte transmission using the RS-232 interface.

Figure 30 shows a typical serial I/O port. The port contains an interface, normally byte wide, to the microprocessor's data bus through which control information and transmit data can be written, and status information and received data can be read. Additional lines coming from the microprocessor include chip select and Read/Write lines plus a few address lines to specify the different internal registers within the chip. The transmit register XMIT is loaded in parallel, one byte at a time, from the microprocessor data

Fig. 30 Typical serial Input/Output Port
bus. The data is then passed in parallel to an output shift register and sent out serially on a single line. Input data are read bit-wise into the input shift register until the register is full. The data are then passed in parallel to the RCV receiver register and from there to the microprocessor bus. The reason for both the receiver and transmitter having two buffers is to allow the microprocessor more time to move data from the receiver before it is overwritten by subsequent incoming bits, plus the ability to fetch the next data to be transmitted in parallel with sending the previous. Buffering is important for achieving good performance and correct operation. Serial data transmission is extremely important when connecting Video Display Units (VDUs) to computer systems, whether they be large mainframes or microprocessor-based systems. It is also the norm in wide area networks where typically the interconnection between network nodes takes place via telephone lines. Local area networks, in the main, also use serial transmission, albeit at a much higher transmission rate than the previous two examples. We will briefly meet with some examples of local area network chips in the next section of these notes.

Perhaps the best known convention, or protocol, for serial data transmission is character, usually byte, asynchronous transmission. Very often this is called start-stop transmission. Successive characters appear in a data stream at arbitrary times. Each character is a 'separate message'. Within a character, bits are transmitted and received at a fixed clock rate. Figure 31 shows timing for a single bit. The idle state is assumed to be high. Each character begins with a low or 0 state followed by up to eight data bits, an optional parity check bit, and one or more stop bits. A bit interval is a fixed period of time governed by the local clock in the transmitter and in the receiver. Common clock frequencies are 300, 600, 1200, up to 19.212 kHz. Stop bits are assumed to be represented by a high state. Stop and start bits serve a very important purpose. Obviously they identify the beginning and end of a character, but more important they permit the receiver to synchronize his local clock to each new character. Remember, even if the transmitter and receiver clocks have the same frequency they may be out of phase. The purpose of the start-bit edge is to tell the receiver when to start sampling. The receiver will try to sample in the middle of successive bit times, thereby compensating for any slight difference in frequency between its local clock and the transmitter clock.

The most common interface using the asynchronous serial protocol is called the RS-232 standard. The standard was originally conceived to foster data communications on public telephone networks. The full interface is thus designed to attach to a modem, a device for transforming digital signals back and forth to analog signals which are suitable for transmission over phone lines. However, the standard rapidly came to be used for the attachment of equipment to computer systems even when no remote access and intervening modem were necessary. For these purposes a very restricted part of the full set of interface lines is used — normally, just the electrical standard on the receive and transmit lines.

Fig. 31 Timing for byte asynchronous serial transmission: St = start bit; Sp = stop bit; P = parity.
Many manufacturers make LSI chips to support asynchronous transmission using the RS-232 standard. The generic name for this type of chip is a UART, (Universal Asynchronous Receiver-Transmitter). Figure 32 shows a simplified diagram of a generic UART. The transmission/reception clock frequency, the number of stop bits, the absence or presence of a parity bit, and several other features can be programmed by use of internal control registers within the chip.

![Fig. 32 Simplified diagram of a generic UART](image)

9. LOCAL AREA NETWORKS

Local area networks (LANs) are fashionable, fast moving, and exciting. LANs connect computing equipment at quite high speeds, 1 to 10 M bps, over distances of a few kilometres. They are a natural and essential ingredient, together with wide area networks, for implementing the distributed computing systems so much in vogue today. Given the presentation, (at this School) of lectures by Roland Rosner on networks and by Ian Williers on personal computers and work stations, I have neither the wish nor the need to explain LAN technology and its importance in today's world. However, what I do want to do is to briefly mention some of the special-purpose chips which are available for connecting microprocessors and other systems to LANs. I will focus my attention on one particular type of LAN, namely Ethernet. Ethernet is the LAN standard which appears most advanced in its specification and implementations. It is also the LAN for which a number of manufacturers are just now starting to release VLSI chip sets.

Ethernet was originally pioneered by Xerox in the 70's. The ideas have more recently been taken up by a number of large manufacturers, and international standards have been formulated by IEEE and ECMA. Ethernet, in its most common form, comprises a coaxial cable to which multiple devices or stations can be attached by tapping into the cable. The attachment normally occurs using a so-called transceiver (see Fig. 33). Each station has
Fig. 33 An Ethernet transceiver

a unique address. Information is passed across this shared channel, bit-serially, as packets or frames. Access to the channel by stations wishing to transmit is co-ordinated in a distributed fashion by the stations themselves, using a statistical arbitration scheme. The technique is basically simple. Before attempting a transmission a station checks to see if anyone else is using the cable. If so, the transmission is deferred until the medium is in a quiescent or free state. When this condition is finally met a transmitter can commence sending a frame. However owing to the finite length of the cable, two stations could see simultaneously that the transmission medium is free and both could commence sending at the same time. A station therefore must both send and listen. If it detects the presence of more than one signal level on the cable, it aborts its transmission, waits a random back-off time, and tries again. This technique is called CSMA-CD (Carrier Sense Multiple Access with Collision Detection).

Figure 34 shows the format of an Ethernet frame. Let us first discuss the role of the preamble. You will recall in our previous discussion about serial asynchronous transmission that one had to take care to synchronize the local receiver and the transmitter clocks. The start bit not only specified where a byte started but, more important, helped to make sure that the received bit-stream was sampled in the middle of each bit period. There was also a parity bit to aid error detection and a stop bit to specify where the 'byte message' ended. Sending an Ethernet frame necessitates consideration of similar points even though we are now sending bytes synchronously at a well-defined rate instead of one at a time (asynchronously). One difference we meet immediately is the way in which the clocking of data is arranged at the receiver end of the transfer of a frame between two stations. Instead of there being separate receive and
transmit clocks, the transmitter sends both the data and a clock. This is done on a single information channel, the Ethernet cable, using a technique called Manchester encoding, shown in Fig. 35. The preamble consists of a total of 8 bytes, 62 bits of alternate '1's and '0's which allow clock synchronisation to be established followed by 2 '1' bits which mark the end of the preamble and the beginning of the frame proper. The next fields are the destination and source addresses; both of these are 48-bit quantities. A station recognizing a destination address as its own will accept the frame currently being broadcast on the cable. If the address does not match, then the frame will be discarded.

In addition to specific station addressing, frames may be sent in a multicast addressing mode that is targeted to more than one destination. The source address is
necessary so that a receiver knows which station on the network the frame has come from. Note that a station can send packets to itself — this is a very useful diagnostic tool.

The type field is a 2-byte field which can be used to qualify the meaning of the frame in some way. For example, it could give the number of bytes present in the data field or identify one of several different types of frame. The data field, which can be between 46 and 1500 bytes long, is followed by a 4-byte check sum. This is a 32-bit cyclic redundancy check which is computed and appended by the source, over all fields except the preamble, and checked by the destination to determine if any errors have been introduced during transmission.

We can see that there is a fair measure of complexity in communicating over Ethernet. We can summarize the requirements of any station, as discussed so far, as three groups of functions:

Type A functions
- Tapping into the Ethernet coaxial cable
- Transferring data from a station to the cable
- Transferring data from the cable to the station
- Indicating when a collision takes place, i.e. more than one station is trying to use the cable.

Type B functions
- Manchester encoding and decoding
- Sensing when there is traffic on the cable (carrier sensing)

Type C functions
- Parallel to serial conversion
- CSMA–CD management (link access, preamble generation and recognition, collision handling, etc.).
- address recognition
- CRC generation and checking
- Packetization; data encapsulation and decapsulation
- Buffering

Figure 36 summarizes pictorially these three types of functions. Type A functions are carried out in the so-called transceiver — the coupling mechanism between the station and the coaxial cable. Type B functions are carried out in the box marked encoding and decoding. Type C functions get implemented in the third box, which I have called the link controller — sometimes called LANCE (Local Area Network Controller for Ethernet). Also shown is an interface to the system bus of a microprocessor host. Usually the transceiver is physically separate from the other functions which are typically packaged on a single printed-circuit board. It is interesting to relate the three types of function A, B, and C to the well-known ISO model and the IEEE 802 standard. This is done in Fig. 37.

The complexity of connecting to Ethernet has led to the manufacture of special Ethernet chip sets which implement, in silicon, all the functions A, B, and C. At the
moment, chips are just becoming available in reasonable quantities for functions B and C. This is going to decrease the problem of interfacing equipment to Ethernet, and one also expects the cost to drop. For the moment the provision of a chip for the transceiver functions seem a little further away. However, at least one manufacturer is promising delivery of such a chip by the end of the year. I conclude this discussion of Ethernet by presenting, in Fig. 38, a simplified picture of an interface between the system bus of a microprocessor system and the Ethernet cable, using two special-purpose chips which implement type B and C functions. The chips concerned are from SEEQ. Semiconductor firms such as AMD, INTEL, National, and others are also just now commencing to market similar products.

Fig. 36 Interfacing to Ethernet

Fig. 37 The ISO model, IEEE 802 and low-level functions required to interface to Ethernet. Many of the functions A, B, and C are being implemented in silicon.
10. VIDEO DISPLAYS

Computer graphics is a very important topic at the moment, and in most people's opinion is likely to remain so. I want to cover very briefly a small corner of the technology — that associated with Raster Cathode Ray Tube (CRT) displays. This type of display is currently very popular and appears on most personal computers and professional work stations in one form or another. For further reading on computer graphics, interested readers are referred to the excellent book by Andy van Dam, a many-times lecturer at CERN Computer Schools.

The CRT displays we discuss here are called raster displays because a picture or text is produced on a screen by a beam of electrons which scans a uniform pattern of closely spaced horizontal lines. The screen, which is covered with a phosphor, glows when the electron beam hits it. Pictures are formed on the screen as the beam turns off and on and sweeps across its face. Figure 39 shows the basic principles of a CRT and how a raster scan pattern is created. Note how the rising portion of the sawtooth waves applied to the horizontal and vertical deflection system sweep the beam left to right and top to bottom. During the rapidly falling portion the beam retraces to the left and top. For the duration of the retrace the beam intensity is greatly decreased so as to be invisible. The beam is thus said to be blanked on retrace. To avoid flickering, the screen is refreshed at 30 to 60 Hz. An ordinary TV set is the most familiar raster graphics device you will have met.
Colour displays work in a very similar way except that there are three electron beams, and the screen, instead of being uniformly coated with a single phosphor, is covered with triads of red, green, and blue phosphors. Each of the three electron beams illuminates only phosphor dots associated with one of the three colours. This is arranged by using a shadow mask as shown in Fig. 40. If the red, green, and blue dots are equally illuminated you see a white dot. By varying the relative intensities of the three beams, any colour you want can be produced. Getting an electron beam to strike only the intended phosphor dot and not an adjacent one is a complex and difficult manufacturing task. The
highest resolution display around at the moment, which seems to be only a prototype, is 1700 (horizontal) × 1280 (vertical) dots. A typical low-resolution display has 300 × 200 dots.

We now consider how numbers and letters are typically drawn on a computer display. Conceptually one divides up the CRT screen into a number of picture elements, or pixels, as shown in Fig. 41. Each pixel corresponds to a particular position of the electron beam as it is raster-scanned over the screen. Associated with each pixel is certain display information, namely the relative intensity of the electron beam, or, in the case of a colour display, the three individual electron beams. We can see that the amount of information may be simple: an on/off bit for the single beam of a monochrome display. However, for the production of many different colours we may end up having to provide several bits of intensity information for each of the three electron beams that illuminate the red, green, and blue phosphors. Information about each pixel is stored in a RAM refresh buffer. It is used to modulate the electron beams as they scan over the display screen. A 0- or 1-bit map for every pixel, which just switches the intensity of a monochrome display off and on, is shown in Fig. 42.

The information per scan can be quite large. In the following, let us assume a relatively modest 300 × 200 pixel array. If each pixel has four bits of information associated with it and the screen is scanned at 60 MHz then the data transfer rate from RAM to the display is 1.8 Mbytes per second. Moving this data with a microprocessor is a regular, heavy, and in some cases impossible load, even if a DMA chip is used (see later). The usual arrangement these days (see Fig. 43) is to use a special-purpose CRT.
controller chip which shares access to the refresh RAM with the microprocessor system bus. The controller continuously generates the correct sequence of addresses for accessing pixel information, and in addition produces pulses which can be used to synchronize the raster scan for the CRT. Various types of scan can be programmed by loading registers internal to the chip. Some CRT chips also contain facilities for a cursor output and a light-pen input.

A cursor is a movable pointer displayed on a CRT screen. Usually the cursor is a special character; sometimes it flashes. CRT controller chips supporting cursor control functions can be programmed to maintain the cursor co-ordinates internally. Each time the CRT scan reaches the cursor position a special signal is output which can be used to modulate the normal video drive for the CRT, for example to produce inverted video. A light-pen is a photosensitive device, used to point at a CRT display, which emits a signal when it is illuminated. The pen sees a sharp burst of fluorescent light during the time the electron beam is bombarding the phosphor. It is not sensitive to the more prolonged phosphor glow or to room lighting (see Fig. 44). The output pulse coming from a light-pen is fed into the CRT controller chip, which then copies the current address being output to

![Diagram](image-url)
the refresh memory; an interrupt can also be generated. This value can then be read back from the chip by the host processor.

You will notice that considerable amounts of refresh memory are required if one is to store information on a pixel-by-pixel basis. Our previous example of a 300 x 200 pixel array with 4 bits per pixel would require 30 kbytes of memory. Although the cost of memory has declined and is continuing to do so, many systems use techniques to limit the size of the refresh memory. One method, applicable when only alphanumeric characters are required to be displayed, is to store information on a character-by-character basis instead of pixel-by-pixel basis. Suppose each character is represented by an 8 x 8 array of pixels, then the amount of storage required for a single character if each pixel is individually stored is 32 bytes, assuming 4 bits per pixel. This can be reduced by storing information on a character-by-character basis; usually the ASCII character code in one byte plus a second attribute byte which contains colour, intensity, and other information (such as font type) about the character. Pixel-by-pixel information is then generated from the character information using pre-programmed character generator chips. With this scheme our 32 bytes per character is reduced to 2 bytes. A page of 25 lines of around 60 characters could occupy only 2 kbytes. It is easy to keep several pages of characters in the display at the same time, and switch backwards and forwards rapidly or even scroll between pages. One further technique to reduce the size of the display memory, whilst retaining full graphics capability, is to allow only a restricted set of colours to appear at any time: n bits of pixel information could point to $2^n$ separate 'palettes' or colour maps, each containing, say, $2^m$ bits of further, more detailed, colour information where $m > n$ (see Fig. 45).

![Fig. 45 Use of colour 'palettes']

11. **KEYBOARDS**

A keyboard is by far the most common means by which a user of a computer system enters information. Most of you will be familiar with its external appearance and use; in this section we look a little at keyboard internals.

Figure 46 shows a very simple keyboard of 16 keys driven from a microprocessor system bus via parallel I/O registers. The keys are arranged as a $4 \times 4$ matrix. In the
Illustration the black key has been pressed and its closure must be detected. To do this,
one uses input and output registers, more normally in practice parts of registers, to
implement a keyboard scan. This is done by a key scanning program activating one column
at a time with the output register, and reading the resultant row lines via the input
register. When a key is depressed it shorts a column to a row. When the column is acti­
vated so is the row. The combination of column output pattern and row input pattern
uniquely identifies the key that has been pressed. Figure 47 illustrates how a keyboard
scan is performed.

People typing on a keyboard often depress more than one key at a time. This is
called 'roll over'. Keyboard scanner programs can take account of more than one closure
and keep track of the sequence in which keys are depressed and released.

---

Fig. 46 A very simple keyboard

Fig. 47 How a keyboard scan is produced
In any situation involving electromechanical components, true contact closure occurs only after an oscillation or 'bounce' period of some milliseconds, usually 10 to 20. Figure 48 shows this effect; note that it occurs when a key is depressed and then released. The scanner program takes this bounce problem into account in the following way. When a key is detected as depressed, the program does not immediately accept that the closure has occurred. Instead, it requires it to be seen as depressed on a number of successive scans before marking the key as being truly depressed. The same happens when a key is released.

For a simple keyboard we have seen how one or more keys can be detected as being pressed and subsequently released. The same principle can be applied to more complex keyboards such as that shown in Fig. 49. We remark that the type of computer system which would use this keyboard would find the continuous load put on it by a scanner program unacceptable. It would have to scan continuously, taking account of debounce, and interpret multiply depressed keys in terms of explicit commands or, more usually, alphanumeric characters. Whilst it is possible to use extra hardware, including special keyboard interface chips, to reduce this software overhead, I want to illustrate a more modern and powerful approach — that of using a subsidiary microcomputer chip to take care of the keyboard. Figure 50 shows such an arrangement very simply. A typical microcomputer chip to use for such an application would be the INTEL 8040 or the Motorola 6502.

![Fig. 48 Contact bounce](image)

![Fig. 49 Keyboard from the IBM personal computer](image)
Fig. 50 Keyboard scan using dedicated microcomputer chip

6530. This type of chip could contain an 8-bit microprocessor, a small amount of RAM around 1 kbyte of ROM or EPROM, a timer, and three I/O ports. The latter are used to manage the keyboard and to send information to the main microprocessor in the system (usually in a serial form). With a little thought you can appreciate the power of such an approach. Not only can the main processor be relieved of the load of continuous keyboard scanning, the subsidiary microcomputer chip could be customized to interpret different keyboard layouts. (Recall that the well-known standard QWERTY keyboard was designed in the 1870's to be purposely so awkward that typists could not type fast enough to jam the keys on a mechanical typewriter.) There are alternatives which are the result of extensive research on the relative frequency and sequencing of characters. You might like to be able to switch back and forth (you would need to change the keyboard overlay!), and this could be implemented in your scanner program running in the keyboard microcomputer. Different character sets might be introduced for different languages, including languages which did not use the Roman alphabet. Any combination of keystrokes can cause any combination of characters. Different characters could be generated by depressing and releasing a key. In fact there are alternatives as to where the interpretation of physical key(s) to character code(s) takes place. It can all be done in the keyboard microcomputer, but on the other hand the keyboard computer could merely pass a key code -- the number of the key plus if it was seen as depressed or released -- to the main microprocessor. The advantage of the latter approach is that whilst off-loading the routine scanning of the keyboard, the main processor is free and is more capable of interpreting keystrokes in a perfectly general and dynamic way than the keyboard processor (which has its program in ROM).

One final comment should be made. You will see that I have treated the topics of CRT display and keyboard separately. Many of you will note immediately that the VDU terminals you use on mainframe and minicomputers are a display and a keyboard packaged in a single box. You may wonder what is the relationship of your VDU to what I have said about bit-map displays and keyboards. Figure 51 shows that what is inside your VDU is in fact just these two components. However, there is a difference: the coupling between the
refresh memory and main processor is weak. By weak I mean that it goes via a slow-speed serial link and is not directly accessible via the host's system bus. This introduces several limitations on graphics-type applications. You can change the refresh memory at the rate of only 1 bit per 100 - 200 µs instead of 16 bits every few microseconds. Hence animation is just not on -- you cannot play PACMAN on your VDU. It is only set up to display characters, and in any case the rate at which you could change the picture would not make for a very exciting game.

12. OF MICE AND MEN

Like many of the ideas behind the present generation of personal computers and professional work stations, the digital 'mouse' has its origins in Silicon Valley back in the 60's. The idea behind the mouse is to provide a convenient mechanism for a user to point to different parts of a CRT screen and to signal to the computer responsible for running the display that some action is necessary. The philosophy behind the mouse and other 'point and pick' devices such as light pens is to provide a more friendly interactive interface between computers and their users. In systems using a mouse-type of device the role of a conventional keyboard is de-emphasized. For example, instead of having to type a series of commands to initiate a program, a user would be given a menu of possible choices containing not just alphanumeric characters but also small pictures called icons. Icons represent some concept or object; they can tell you what they represent better than words alone. A user would then select an option by simply pointing to it. Figure 52 shows a display generated by an Apple Macintosh which illustrates the
concepts of menus, icons, and choosing through pointing. As I do not want to steal material from Ian Williams I will say no more about the display side of things, but instead go on to explain how a simple form of mouse is constructed and interfaced.

The first mouse, circa 1964, was a 2 x 3 inch block of wood. It had three push buttons on top for attracting the attention of the computer to which it was connected. The inside was hollowed out to provide room for two perpendicular wheels on which it moved over any flat surface. Each of the wheels was attached to the shaft of a separate potentiometer. You will recall that a potentiometer is just a variable resistor. In this case the variable resistors can be used to generate variable voltages, which could then be measured and digitized using ADCs and used to indicate the mouse's relative movement in two orthogonal directions. The first mouse used with the Xerox Alto work station in the early 1970's actually employed this perpendicular wheel concept although the potentiometers were replaced by digital encoding.

A variation of the original idea incorporated a ball 'rolling' as the mouse was moved back and forth across a surface. The ball transmits its movement to two perpendicular wheels. The function of the ball is to smooth out movement of the mouse during angular travel. Figure 53 shows a CRT raster display picture of a mouse. More recent developments have considerably refined mouse technology, and a variety of products, some including embedded microprocessor chips, are available on the market.

Although any discussion of this topic has of necessity been brief, I hope you will at least appreciate the basics of how a mouse works and the importance of it and similar devices in man-computer interactions. There is a trend away from the keyboard and this may help the many people that "could not" type anyway!
13. PRINTERS

There is a wide range of printers available for connection to microprocessor systems, and there are several ways to categorize the different types. One can classify them according to how printed characters are formed and also the technical methods used to print a character on paper. Printed characters are usually formed in two ways: by a dot matrix, and as fully formed characters. With a dot matrix technique, characters are created by a series of dots as shown in Fig. 54. Usually a $5 \times 7$ or $9 \times 9$ dot matrix is used. The matrix can be used to form all the letters of the alphabet (both upper and lower case letters), numbers, punctuation marks, and special symbols. Matrix printers can also draw pictures, charts, and graphs. Several different character sizes and fonts may be possible. Fully formed characters are like those on a typewriter. The character is formed of metal or hard plastic and causes a smooth, continuous character to be printed. This method gives high quality print. Printers working in this way are generally known as letter-quality printers. Most letter-quality printers are of the 'daisy-wheel' type. A daisy or print wheel is a circular metal or plastic dish-like device made up of 96 flat, leaf-like petals around a circular hub. Each petal is a fully formed character. The printer rotates the daisy wheel to the selected character and a print hammer bangs it against the ribbon and paper. Daisy-wheel printers give a higher quality print compared with matrix printers. However, they tend to cost more, print more slowly (20 characters per second compared to 80 characters per second), and have no possibility for graphics output.

Most dot matrix printers, like letter-quality printers, use an impact method for printing. The dot matrix pins hit an inked ribbon which in turn impacts the paper. This scheme is shown in Fig. 55. The main advantage of this type of printer is its speed and versatility (graphics and even colour) compared with daisy wheel models.
Interfacing a printer is relatively easy using a standard parallel or serial I/O chip. There is even a standard connection between printers and computer systems in the form of the Centronics Parallel Interface. This interface can be simply built from a handful of TTL glue chips. The Hewlett Packard inspired GPIB (General-Purpose Interface Bus) is another possible connection scheme for printers as well as for a whole range of laboratory instruments.

14. FLOPPY DISKS

The most widely used secondary storage medium for microprocessor-based systems is the so-called floppy disk. I intend to describe the basic operation of this type of device and to ignore other magnetic (and optical) mass storage on the assumption that it will be covered in the lectures by Canon.

Floppy-disk drives record information magnetically in bit-serial form on the coated surfaces of thin platters called diskettes. The coating is the familiar brown metal oxide you find on tapes for home recorders. The diskette is permanently enclosed in a square cardboard envelope lined internally with a low-friction material. When in use the 'floppy' rotates at high speed inside its cover while the Read/Write heads of the drive are pressed against the magnetic surface. Data is recorded on the disk surface in a series of concentric circles called tracks, and each track is further divided into equal-size pieces called sectors. One moves from one track to another by shifting the Read/Write heads closer to or further away from the centre of the spinning diskette. Figure 56 shows a diskette.

Let us now look in a little more detail at the storage format of one floppy-disk system, namely that used on the IBM Personal Computer. Note that other formats offering a factor of 2 more are readily available, and that IBM on their new PC AT have introduced floppy-disk drives with a factor of 4 more storage capacity on the same size diskette.
Square protective cardboard jacket.
Head aperture. Hole to expose disk for contact with read/write head. (One on each side)
Write protect notch. If left open, disk can be recorded on. If covered, usually with tape, disk cannot be recorded on.
Hole in diskette for drive spindle.
Circular mylar plastic disk coated with magnetic oxide on both sides.
Index hole in disk used as reference location for storage format.
Circular mylar plastic disk coated with magnetic oxide on both sides.
Hole in cardboard jacket.
Figure 56 A diskette

Figure 57 shows the format of our diskette. There are 40 tracks on a 5 1/4" diameter platter. The track density is 48 tracks per inch. Each track is divided into 8 sectors, each of 512 bytes plus some control information. Each diskette can therefore store $40 \times 8 \times 512 = 163840$ bytes per side or 320 kbytes total. Data are recorded using a technique known as Modified Frequency Modulation (MFM), which I do not propose to go into but which is described in Stone's book (see list of references). Before use, a diskette must be initialized in a special way (formatted). During this process, space is allocated and marked out for each sector, and a sector ID (address) together with additional control information is written within the sectors. The sector also contains a field for a CRC checksum. Whenever data are written to disk, a CRC is calculated and stored in the appropriate field within a sector. Whenever data is read back, a new checksum is calculated and compared with the CRC on the diskette for that sector. A small hole near the centre of the diskette identifies the start of sector 1, and it is detected as the diskette passes between a LED and a photodiode. This so-called index hole provides a physical reference point on the diskette. The format scheme just described is referred to as soft sectoring.

Special-purpose floppy-disk controller chips are available to simplify the connection of a drive to the system bus of a microprocessor system. A much simplified diagram of a typical controller, which is capable of looking after four drives, is shown in Fig. 58. A number of basic commands can be sent to the controller, which carries out the action required, possibly passing data back and forth between the system bus and the drive. At the end of the action status and other housekeeping information can be read back from the controller. Typical commands include: move Read/Write head of selected drive to a selected track; transfer sectors of data; scan sectors for specific patterns; format a track; sense drive status, etc.
40 CONCENTRIC TRACKS OF DATA (0-39)

INDEX HOLE MARKS
BEGINNING OF SECTOR 1

SPINDLE ACCESS HOLE

ONE OF 8 SECTORS

512 bytes of data stored in each sector per track

40 tracks × 8 sectors × 512 bytes per sector = 163,840 bytes (160K) per side.
2 sides = 163,840 × 2 = 327,680 bytes (320K)

Fig. 57 Format of a diskette

Fig. 58 A disk controller chip
15. DMA CONTROLLERS

In much of our previous discussion we have assumed implicitly that the microprocessor is in control of data transfers between memory and I/O devices. We have, however mentioned that other sources of control could coexist with the microprocessor chip. Recall the use of arbitration lines to facilitate this. We now consider the role played by one type of special-purpose processor: the direct memory access (or DMA) controller.

Suppose we consider the transfer of data between an I/O device and memory. If we code this using a typical sequence of microprocessor instructions we would get something along the following lines. (The number of bus cycles appears afterwards in parenthesis.)

1) Read status of interface (2 cycles to fetch instruction + 1 I/O cycle).
2) Is interface ready? (2 cycles to fetch instruction).
3) If not, go to 1 (2 cycles to fetch instruction).
4) Input data (2 cycles to fetch instruction + 1 I/O cycle).
5) Store data in memory (2 cycles to fetch instruction plus 1 memory cycle).

This gives a total of 13 cycles of which any 2 actually move data between the I/O device and memory. Clearly some devices will be too fast to use this type of programmable I/O, which will give an overhead of around 10 μs per data word. In other situations where a microprocessor might be able to keep up with the required speed, the load on it could, in many cases, be unacceptable. The answer is to build special-purpose circuitry that is designed to transfer blocks of data at high speed between memory and I/O devices. Such a circuit has the same capability as the CPU chip to generate addresses, timing, and command signals. It is called a DMA controller. A DMA controller essentially executes a block move instruction: MOVBLOCK source, destination. DMA chips are found in all but very low end microprocessor systems. Figure 59 shows how DMA capability can be added.

Normally the processor will have control of the bus, and the DMA controller must acquire bus mastership whenever a transfer is to take place. Unless both the source and the destination are capable of supplying or accepting data very rapidly, say in the case of a memory-to-memory transfer, then either the source or the destination must initiate or trigger each transfer. This necessitates some signals between the synchronizing device and the DMA controller. These signals are a DMA request (DMAR) to the controller and, optionally, a DMA acknowledge (DACK) from the controller to the device. We note in Fig. 59 that additional signals, the HOLD and HOLDA discussed previously, are also required in order to allow the DMA controller to take over control of the bus from the microprocessor. Let us now look at different types of DMA transfer.

Once the DMA controller has acquired mastership of the microprocessor bus from the CPU, in order to carry out the transfer of a block of data, one asks how long may it retain it. Various possibilities exist. The controller could retain the bus for the duration of the whole block transfer, or at least as long as data were available to be transferred. This clearly maximizes the overall data transfer rate because the overhead in acquiring the bus is amortized over the maximum number of bus transfer cycles. However, this somewhat antisocial behaviour may introduce an unacceptable 'latency of access' for other bus users. A more usual mode is for the controller to request use of
the bus when data are ready to be transferred and relinquish mastership after one or a few transfers, or when there are no data ready to be transferred — whichever happens first. We must also consider how data actually flow over the bus during a DMA transfer. One solution is to use a two-cycle DMA and buffer the data in the controller. During one bus cycle the DMA controller accesses the source, storing it in an internal register. In a subsequent cycle the data is written to a destination address. An alternative solution allows a single-cycle DMA transfer between an I/O device and memory. You may wonder how this can be arranged, given that only one address at a time can be present on the address bus. However, the answer lies in the use of the DACK line, which is sent from the DMA controller to the I/O device asserting DMAR when a transfer is taking place. This line is used as an alternative addressing mechanism to select the I/O device. The DMA controller uses the normal address lines to specify the memory location involved in the transfer. Timing is most conveniently arranged by the use of separate I/O and memory strobe signals. One bus cycle can then transfer information directly between an I/O device and
memory. This type of transfer is often called a 'fly by' transfer, as information flies by the DMA controller and is not stored within it. Figure 60 illustrates single- and double-cycle DMA.

![Diagram of single- and double-cycle DMA]

Fig. 60 a) Double and b) Single DMA cycles

What would one find in a DMA controller chip? In addition to the circuitry needed to obtain access to the microprocessor bus and to drive it, there are programmable registers to regulate the DMA operation. One register is needed to specify the address of the source and another for the destination. A third register is needed to program the number of transfers required to take place. Other registers are used to define the type of DMA transfer to be carried out. What is the algorithm for applying for bus mastership and for releasing it? Is the transfer one or two cycles? Will the source and destination address be incremented, decremented, or left alone after every transfer?

Most DMA controllers are designed to handle several independent channels -- two or four is usual number. Each channel has its own source, destination, count, and configuration registers. In this way multiple transfers may be active simultaneously, although only one channel has control of the bus at any one time.

One can identify several other types of device that exhibit DMA controller-like behaviour. One very natural evolution of the standard DMA controllers we have discussed so far is to a so-called smart DMA controller or I/O processor. Along with being able to perform conventional DMA operations, an I/O processor is endowed with the ability to execute its own specialized instruction set. These instructions are fetched from memory and executed in an analogous way to standard microprocessor chips. In addition to merely moving blocks of data around, an I/O processor can be programmed to perform a large variety of other useful jobs: for example, data acquisition, data buffering and
conversion, link protocol management (error checking and re-try), intelligent disk controller, and so on. Figure 61 shows a typical application for an I/O processor. An INTEL 8089 acts as a front end to an INTEL 8086 microprocessor. The 8089 executes its own program under the direction of the 8086. Communication between the two processors is via tables in a shared memory which can be accessed by both parties. The 8089 can relieve the 8086 of much of the I/O load and can intelligently transfer blocks of data at much greater speeds than are possible with the CPU alone. It is interesting to note that the Ethernet chips now becoming available from AMD, INTEL, and others embody within them many of the characteristics of an intelligent I/O processor. We can look for still more intelligence in support devices in the future.

![Diagram of I/O processor](image)

**Fig. 61 Use of an I/O processor**

16. FLOATING POINT PROCESSORS

Standard microprocessors rarely provide for execution in hardware of floating-point instructions. Calculations which require floating-point arithmetic must be done in software, either by subroutine calls or via emulation of floating-point instructions. Several manufacturers provide, or plan to provide, floating-point processors to be used in association with normal processor chips. In this section we give some a few details of one such floating-point processor chip and present some performance comparison figures for operations with and without floating-point hardware.

Figure 62 shows how an INTEL 8087 can be used together with an 8086 microprocessor. As far as a programmer is concerned, he or she is only aware of a single machine.
However, the 8087 adds new data types, new 80-bit registers, and extra instructions over and above those normally present in the 8086. The 8087 acts as a co-processor. It watches as instructions are fetched by the 8086, monitoring the main processor status lines to track exactly what the 8086 is doing at any time. It processes its own instructions, ignoring those belonging to the 8086. In a complementary way the 8086 processes its instructions whilst largely ignoring those of the floating-point processor. The 8086, however, does perform one important service for the 8087. On seeing a floating-point instruction the 8086 calculates any effective address required and makes it available to the 8087. The 8087, after obtaining local bus mastership from the 8086 and using a HOLD/HOLDA mechanism, can use the address to fetch and store operands. The main processor then proceeds to the next instruction, leaving the floating-point processor to work in parallel on its calculations. Also shown in Fig. 62 are two pins, BUSY on the 8087, and TEST on the 8086. These allow the 8086 to test and wait on the termination of a floating-point instruction in the cases where this is necessary: for example, if a second floating-point instruction is sensed or if the 8086 wants to use an operand currently being used by the 8087. The 8087 is also provided with a mechanism to interrupt the 8086 in the case of invalid operations and exceptions; divide by zero, square root of a negative number, and so on.

This has been a brief treatment of a complex and interesting topic. We have seen how a main CPU can generate on its bus an instruction stream which consists of a mixture of CPU and co-processor instructions. The co-processor automatically selects its own instructions and acts on them in parallel with the main processor. Figure 63 illustrates

<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>APPROXIMATE EXECUTION TIME (MICROSECONDS)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8087</td>
</tr>
<tr>
<td>ADD SIGN-MAGNITUDE</td>
<td>14</td>
</tr>
<tr>
<td>SUBTRACT SIGN-MAGNITUDE</td>
<td>18</td>
</tr>
<tr>
<td>MULTIPLY (SINGLE PRECISION)</td>
<td>19</td>
</tr>
<tr>
<td>MULTIPLY (DOUBLE PRECISION)</td>
<td>27</td>
</tr>
<tr>
<td>DIVIDE</td>
<td>39</td>
</tr>
<tr>
<td>COMPARE</td>
<td>9</td>
</tr>
<tr>
<td>LOAD (DOUBLE PRECISION)</td>
<td>10</td>
</tr>
<tr>
<td>STORE (DOUBLE PRECISION)</td>
<td>21</td>
</tr>
<tr>
<td>SQUARE ROOT</td>
<td>36</td>
</tr>
<tr>
<td>TANGENT</td>
<td>90</td>
</tr>
<tr>
<td>EXPONENTIATION</td>
<td>100</td>
</tr>
</tbody>
</table>

Fig. 63 Performance of the 8087 floating-point processor
the power of this approach by comparing the time for some typical floating-point operations on the 8086 with and without the use of the 8087 co-processor. For further reference, a floating-point multiply on a CDC Cyber mainframe takes 1 μs instead of 19 μs. It does cost a bit more though!

17. DIGITAL-TO-ANALOG AND ANALOG-TO-DIGITAL CONVERTERS

Microprocessors find wide use in many areas of data acquisition and control, including all sorts of low end systems for everyday living, e.g. domestic appliances, phone systems, games, and so on. We briefly review what is meant by data acquisition and control, and explain the importance of converter chips that transform between the analog real world environment and the digital (ones and zeros) world of computer systems.

Figure 64 summarizes very simply what is meant by data acquisition and control. The term 'process' as used in our discussion is taken to mean any real world happening. It could be building a car, making a pot of tea, or producing a new elementary particle. It is a very general term! One learns about the process by acquiring data from it, and one controls the process by varying some external parameters which influence it. Before the days of computers, data acquisition took place manually by observing and recording the readings of meters, dials, and gauges; control took the form of human manipulation of voltage levels, valves, switches, etc. Nowadays much of this can be and is automated.

In very few situations is the data you want to acquire an electrical quantity. More often a transducer is necessary to convert a physical quantity into a 'corresponding' electrical form. Examples of the basic physical quantities we want to measure are numerous; temperature, light level, magnetic field, position displacement, and time, to name a few. To convert these quantities to an electrical form we use thermocouples, photodiodes, Hall probes, strain gauges, time interval to charge or voltage converters, and so on. The same problem arises when we want to control or change some parameter. The 'end parameter' is seldom an electrical quantity. Again some form of transceiver will be required, this time to transform electrical energy into some other form of energy. For example, a loud speaker transforms electrical energy into sound energy; a light-emitting diode converts electrical energy to light energy. You can think of many other examples from your own experience.

Even if we can transform physical quantities to and from electrical signals using transducers, we still have the basic problem that these signals will in general be

---

Fig. 64 Simplified representation of data acquisition and control
continuous or analog in nature. Computers represent and manipulate information in digital form, strings of binary bits being either 'zero' or 'one'. Hence we can immediately see the need for, and the importance of, analog-to-digital and digital-to-analog converters. They allow a computer to manipulate information internally in digital form yet process and provide external signals in analog form.

Figure 65 shows the principle of one type of digital-to-analog converter (DAC). It is based on a binary weighted resistance ladder. Each element of the ladder is either connected to ground or to a reference voltage \( V_{ref} \). The state of this connection is conditioned by a digitally controlled switch. If a one is present the switch converts the resistor to \( V_{ref} \). If a zero is present the connection is to ground. For a four-element ladder controlled by a 4-bit number \( b_3b_2b_1b_0 \), where the \( b \)'s are zero or one and \( b_3 \) is the most significant bit, the output voltage is

\[
V_0 = -V_{ref} \left( \frac{b_3}{2} + \frac{b_2}{4} + \frac{b_1}{8} + \frac{b_0}{16} \right),
\]

hence, \( V_0 \) is proportional to the value of the number whose binary representation is \( b_3b_2b_1b_0 \).

DACs have typically a maximum output voltage in a range between -10V and +10V. The resolution varies from 4 to typically 18 bits. Settling times for the analog output vary from a few tens of nanoseconds for ultra high speed devices to some tens of microseconds or more for very accurate converters. Figure 66 shows the pin assignment for a low cost 8-bit DAC. The chip contains an internal voltage reference source. The output range can be chosen to be 0 to 2.5 V or 0 to 10.0 V using external strapping pins \( V_{out} \) \( SENSE \) and \( V_{out} \) \( SELECT \). New data to be converted can be written into a storage latch directly from a microprocessor bus under the control of the two chip select lines \( CS \) and \( CE \). The output settling time is 1 to 2 \( \mu s \).

There are several methods of implementing analog-to-digital conversion. These methods span conversion times from tens of milliseconds to some tens of nanoseconds for
the ultra fast flash ADCs. We define conversion time as the time to perform the digitalization. All converters except the flash ADC are capable of 12-bit resolution or better. The highest-resolution ADC I could find in the Analog Devices databook had a 16-bit resolution and a 50 μs conversion time. Flash ADCs have typically a maximum of 8 bits and a conversion time of 30 to 50 ns. We shall look at two types of ADC conversion techniques followed by one well-known chip.

Figure 67 shows a counter or staircase ramp ADC. At the start of conversion the counter starts counting up from zero. The DAC converts the count into a voltage which is compared to the input voltage. When the two are the same the counter stops counting, and its value, which represents \( V_{\text{in}} \), can be read. The conversion time can be quite slow, in the 10 ms range. Is there no better way than simply counting up from zero? The answer is Yes, you can use a binary search algorithm. In Fig. 67 we replace the simple counter by a successive approximation register (SAR). At the start of conversion the SAR outputs a value with the most significant bit one and all other bits zero. If the comparator indicates that the DAC output is too big, the bit is removed from the SAR, if not it remains. The next significant bit is then set and the procedure repeated until all bits have been tried out. This type of successive approximation can be both fast, 10 μs conversion time, and accurate, 12 bits of data.
Figure 68 shows the popular National Semiconductor ADC0816 chip. It contains an 8-bit successive approximation ADC with a conversion time of 100 µs and also a multiplexer which allows 16 input channels to share, in turn, the ADC. The multiplexer channel can be selected using a 4-bit address. Input control signals are provided for latching the input address lines, for enabling the resulting 8-bit digital output, and for starting the conversion process. An end-of-conversion output signal indicates that valid data can be read back from the chip.

![Diagram of ADC and 16 channel multiplexer](image)

Fig. 68 ADC and 16 channel multiplexer

18. **TIMERS**

Microprocessor systems require information about two types of time: relative and absolute. Relative time marks intervals between events, e.g. generating a refresh cycle every 15 µs, updating a display every 1 s. Absolute or real time is something we are perhaps more familiar with; we say 'it is 10:00 a.m. on June 26th 1983'. Absolute time is used in microprocessor systems for such things as marking files with a time and date of creation and for synchronizing the microprocessor with events in the 'real' world.

Let us look at some basic time-keeping circuits. Figure 69 shows a programmable timer. A precise frequency, which may be derived from the master clock of a microprocessor, is fed into a counter which counts down from some value, programmable from a microprocessor, to zero. When the counter contains zero its output level will change. This happening can be used to generate an interrupt in the microprocessor. Figure 70 shows a slightly more complex arrangement for keeping track of the time of day. The crystal oscillator output is prescaled down to 1 Hz and passed through a series of three counters. The counters count up to 60, 60, and 24, respectively, before giving an output pulse and resetting automatically to zero. The counter contents can be set to some initial time and thereafter read to determine the time of day.

Recognizing the need for good time-keeping, integrated circuit manufacturers applied their silicon prowess to the task. The result has been a good assortment of timer chips. Figure 71 shows the INTEL 8253 programmable interval timer. The interface to the
Fig. 69 Simple programmable counter

Fig. 70 A simple time-of-day clock

Fig. 71 Programmable interval timer
microprocessor bus is fairly straightforward. Dates can be written to and read from internal registers within the chip using an 8-bit bidirectional data bus, two address lines, a chip select line, and Read and Write timing and command lines. There are three separate 16-bit counters and mode registers for programming the mode of operation of each of the counters. Each counter has three pins associated with it. There are two inputs, a clock to drive the counter, and a gate to enable and control counting. The single output may be used to interrupt the microprocessor or drive other circuitry.

There are six possible modes which can be programmed for each counter independently:

- **Mode 0.** The counter counts down from a preset value and asserts its output when it reaches zero.
- **Mode 1.** The counter begins to count down, negating its output as it does so, in response to a transition on the gate input. The output will stay negated until the counter contains zero. At this time the output is reasserted. The counter can be retriggered by another appropriate transition on the gate input. It therefore acts as a re-triggerable one-shot.
- **Mode 2.** The counter continually counts down, reaches zero, and reinitializes. For the short time when its contents are zero the counter output is low, otherwise it is high. In this mode, pulses are produced at regular intervals.
- **Mode 3** is very similar to the previous mode except the counter output is high for the first half of the count and low for the second half. It thus acts as a square waveform generator.
- **Mode 4** is similar to mode 0 except that a short pulse is output after the counter has counted down to zero.
- **Mode 5,** like the previous mode, generates a pulse when a countdown to zero has occurred. However, the external gate is used to start the counter.

One can see the versatility of this timer chip for all manner of applications, including the playing of music via an amplifier and loud speaker. There are several similar devices and also chips which can be used to give real time-of-day information, including optional interrupts once a month, week, day, hour, minute, and second.

19. **PUTTING IT ALL TOGETHER**

In this section we will look at four examples of microprocessor-based systems. Together they cover just part of the wide spectrum of possible applications for this exciting technology. However, I hope you will be able to identify in each case the basic components I have introduced in these lectures.

In 1976 General Motors introduced the first car spark-timing system controlled by a microprocessor. Figure 72 shows the essentials of the system. Sensors supply the microprocessor with the required input information, namely engine vacuum, crankshaft position, reference timing and coolant temperature. Three outputs are produced. The main one is the timing signal. It is gated to the spark plugs via the distributor. The other two are status information 'check ignition' and 'coolant hot'. The microprocessor is a special-purpose device with on chip RAM and ROM and digital and analog I/O lines. The
The major advantage of the system is claimed to be the reduction in engine emission (pollution!) because of the precisely controlled nature of the ignition timing.

Figure 73 shows a microwave oven controller. Control is accomplished using a single-chip microcomputer such as the INTEL 8048. There is a 28-key hexadecimal keyboard which is used as the input mechanism to issue basic commands. A user will specify the time the oven is to start cooking, the nature of what is being cooked, and the weight. If unreasonable data are entered e.g. "chicken", "30 lbs.", an audible alarm is set off and the data are rejected. Provided that the data entered are reasonable, the controller will compute and implement the necessary cooking time. Several other functions are provided,
including a time display, door close and temperature sensing, etc. This area of application is one which has traditionally been carried out in the past by electromechanical methods. Microprocessor control offers the same basic functionality as the older type of control mechanism but with the ability to provide more convenience and features. The 8048 chip, or equivalent, contains — in addition to an 8-bit CPU — RAM, ROM, programmable timers, and I/O lines.

Our third example of putting it all together is a generic personal computer; let us call it the RAINMAG PC. It runs an operating system called DOSNIX-MPC. Figure 74 shows the features of such a PC and we list them below with some further short comments.

- The CPU is either a Motorola 68000 family member or one of INTEL's 8086/8, 186/8, 286 series. The processor is capable of addressing around a megabyte of memory. There may even be two standard microprocessors, each capable of running different operating systems and application programs. In one PC the second processor in fact emulates an IBM 370 mainframe.
- There is an optional floating-point co-processor which probably is not too well supported by the available software.
There is a four-channel DMA controller used for driving the floppy disk and local area network controllers. Perhaps the manufacturer also uses one channel to periodically refresh the dynamic memory.

There is an interrupt controller allowing eight vectored interrupts to the processor.

There is 64 kbyte of ROM in which is contained power up diagnostic software, parts of the operating system, and some useful run-time utilities. Almost certainly there will be a version of BASIC running out of ROM too.

There is a three-channel timer chip which is used by the operating system to keep track of real time and maybe to trigger memory refresh cycles. Users may use the third spare channel as they wish, or the system may perhaps use it for generating audio output (music?) via a loudspeaker.

The amount of memory on the system will be typically 128 kbytes. If the manufacturer is sensible he will allow for more. The memory will most likely be implemented with 64 k x 1-bit dynamic memory although, 128 k and 250 k chips are arriving. Refresh may be implemented at a system-wide level, for example using a DMA controller channel, or at a local, memory level.

Secondary storage is typically provided by two floppy disk drives whose capacity may range from 0.3 to 1.2 Mbytes per drive. The drives are operated using a floppy-disk controller chip.

There is a full-size keyboard and perhaps a mouse to allow the user to interact with his PC. Serial and/or parallel I/O chips are used to interface these devices.

The CRT screen, which may be colour, is a bit-map raster scan device. Its resolution, which depends on the number of colours available at any time, ranges from 160 x 100 with 16 colours to 640 x 200 for monochrome. The refresh memory can be accessed by the processor as part of its memory space and also by a CRT controller.

There is an 80 character per second printer interfaced according to the Centronika standard and capable of producing graphical output—a copy of the picture on the CRT for example.

A connection to a local area network, probably Ethernet or a near relation, has been implemented using one of the newly available protocol chip sets.

A connection to a wide area network could be made via an RS-232 port and a modem. However, it is more likely that such a function is provided by a separate gateway on the local area network.

You hope that there is, or will be, a vast reservoir of useful and cheap software of all types, together with glossy publications containing 'useful' articles on how your PC can be programmed in BASIC to play scrabble, to baby sit, to improve your teenage children's grades, and so on.

If you are lucky or smart in what you buy, then your generic PC comes with clear information as to how to connect equipment to its system bus. You should be able to build or buy boards for a whole variety of applications including data acquisition, plus adapters to standard microprocessor buses and instrumentation buses such as CAMAC and GPIB.
- 303 -

• You will despair of trying to understand in detail how the manufacturer's operating system works, and how to tailor it to your needs.

Our final example (see Fig. 75) is a terminal concentrator for Ethernet. The system is synthesized from MULTIBUS boards. It comprises a processor board containing a CPU, RAM, and PROM, an Ethernet controller board and several boards equipped with asynchronous RS-232 compatible serial I/O ports. Figure 76 shows a MULTIBUS processor board. The processor board is responsible for supervising the packetization of keyboard commands from multiple VDUs and passing them over Ethernet to the appropriate host. It is also

![Diagram of terminal concentrator](image)

**Fig. 75** Synthesizing a terminal concentrator (8 to 16 lines) for Ethernet using MULTIBUS boards

![Diagram of MULTIBUS processor board](image)

**Fig. 76** A MULTIBUS processor board
responsible for traffic in the opposite direction, namely frames sent from different hosts containing information destined for display on multiple VDU screens.

Microprocessor technology has seen hectic and exciting times over the last 15 years and the pace shows little sign of slackening off. Let me close with a quotation from a recent issue of Scientific American. 'If the aircraft industry had evolved as spectacularly a way as the computer industry over the past 25 years, a Boeing 767 would cost $500 today, and would circle the globe in 20 minutes on a gallon of fuel. Such performance represents a rough analogue of the reduction in cost, the increase in speed of operation, and the decrease in energy consumption.' There are still more exciting times ahead!

ACKNOWLEDGEMENTS

It is a pleasure to acknowledge the people who made my first sabbatical year at the U of I so fruitful and provided the stimulating environment in which I have written these lectures — in particular, Dick Brown, Bob Downing, and my students, (Kurt, Mark, and Sheldon) who educated me in many aspects of computing and electronics and who made me so welcome.

Jeannine Adomaitis at the U of I has laboured through the hot Midwest summer deciphering my writing and entering these notes into her word processor. My sincere thanks to her for her hard work and also to Kitty Wakley at CERN for improving my English in her normal, highly competent manner.

My thanks to Emily Indah, my daughter, who since her arrival in the summer of 1983 has helped put the world in general, and high-energy physics in particular, into perspective.

Finally, a sincere thank you to my long suffering wife, Janet, who provided tea, beer, sympathy, etc. at appropriate moments.
There have been hundreds of textbooks written on the subject of microprocessors; still more magazine articles. The semiconductor industry itself publishes numerous and detailed catalogues of individual components. I have listed below just a few of the books I have read and found useful in preparing these lectures.


H. S. Stone, Microcomputer Interfacing (Addison-Wesley Publishing Co., 1982).
