Microprocessor

Data Book

SECOND EDITION

S. A. Money

Microprocessor Data Book

SECOND EDITION

This page intentionally left blank

Microprocessor Data Book

SECOND EDITION

S. A. Money

ACADEMIC PRESS, INC.
Harcourt Brace Jovanovich, Publishers
San Diego New York Boston
London Sydney Tokyo Toronto

This book is printed on acid-free paper. ∞

Academic Press, Inc.
San Diego, California 92101

United Kingdom Edition published by
Blackwell Scientific Publications Limited
Osney Mead, Oxford OX2 0EL, England

Library of Congress Cataloging-in-Publication Data

Money, Steve A.
Microprocessor data book / S. A. Money. -- 2nd ed.
p. cm.
Includes bibliographical references.
ISBN 0-12-504445-3 (alk. paper)
1. Microprocessors. 2. Microcomputers. I. Title.
QA76.5.M549 1990
004.16--dc20 90-300
CIP

Printed in the United States of America
90 91 92 93 9 8 7 6 5 4 3 2 1

CONTENTS

PREFACE

Advances in the techniques for manufacturing large scale integrated (LSI) circuits have, in recent years, made it feasible to incorporate most, or in some cases all, of the complex logic required for a small digital computer system on to a single silicon chip. One example of the application of these LSI techniques is in the familiar digital pocket calculator, which is in fact a specialised digital computer. In these devices all of the electronic logic is contained in a single integrated circuit package.

In designing modern electronic systems the engineer must now take into account the ready availability of microcomputer and microprocessor devices which can simplify design, making the end product more versatile or more economical to produce.

One problem which faces the designer planning to use a microprocessor is the great multiplicity of devices that have become available. Choosing a suitable microprocessor could involve collecting together and searching through a mountain of different data sheets and manuals.

In this book condensed data have been provided for most of the available types of microprocessor and microcomputer device. For each major type or series a description is given of the internal architecture, instruction set, main electrical data and package details.

Most of the popular devices are manufactured by several different suppliers, and a list of alternative sources and type numbers have been included in the data for each type. Support chips designed for that processor have also been listed.

For convenience the devices have been divided into groups covering 4, 8 and 16-bit types and other processors. It would not be practical to include full details of each type, but it is hoped that sufficient information has been provided to allow the designer to narrow down his choice to perhaps one or two types. The manufacturer's data sheets or manuals may then be consulted for more detailed operating and application information.

In order to choose a processor for a project some knowledge of the basic principles of the devices is required, and this has been covered in the introductory chapter. A general guide has also been included on the factors involved when a processor type is chosen.

A complete system normally consists of a microprocessor together with a selection of supporting devices to handle input–output, external device control and to provide memory. The number of support devices available is even greater than that of microprocessor types, so no attempt has been made to include details of all of these. Some descriptions have been included covering the major support device functions, and data have been included on some of the more popular types as a guide to the facilities provided by such devices.

At the end of the book a directory of microprocessor manufacturers has been included and there is also a glossary of some of the terminology used in the microprocessor field.

It is hoped that the information given in this book will assist designers in choosing suitable devices and that it will be generally useful to those engaged in designing or planning microprocessor based products.

One problem encountered in producing any data book which deals with a rapidly advancing field, such as microprocessors, is that new devices are continually being introduced. To deal with this situation plans are being made for the publication, from time to time, of a supplement giving data on recently introduced devices. Readers wishing to have details of these supplements should complete and mail the coupon enclosed in this book, or alternatively write to the publishers, when they will automatically receive advance details of these supplements.

The reader will notice that for a limited number of devices in this book only limited data are given. This is because at the time of compilation only preliminary information was available. The reader is referred to the supplement for full information.

Finally, I would like to express my thanks to all those manufacturers and distributors who supplied the data and other information which made it possible to compile this book.

1 INTRODUCTION

This page intentionally left blank

In recent years the advent of microprocessors and microcomputers has revolutionised the whole process of digital system design. Projects which, a few years ago, might have required tens or hundreds of digital logic devices can today be implemented by using perhaps one or two LSI circuits. Of course LSI circuits have been around for some years, but economic considerations have usually limited these to applications, such as digital calculators where high volume production is possible and high design costs can be recovered quickly. The advantage of the microcomputer is that a standard device can be used for many applications merely by altering the program of instructions held in its memory. Thus design costs can be reduced and a variety of products may be built using perhaps a standard circuit board.

Microcomputers, however, bring with them a number of new design concepts which may be unfamiliar to the system designer used to working with conventional digital logic systems. In this introductory section we shall examine the internal organisation of microcomputer systems and their general principles of operation. Later we shall consider the various factors involved in choosing a suitable type of microprocessor for a design project.

ARCHITECTURE

The general organisation of a digital computer, whether it be a mainframe, a minicomputer or a microcomputer, follows the basic arrangement show in fig. 1.1.

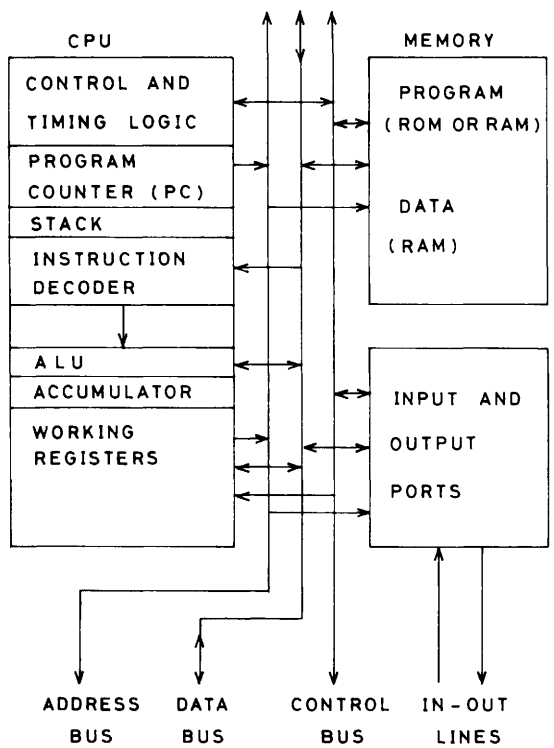

Fig. 1.1

At the heart of the system is the *central processor unit*, generally referred to as the CPU. Functionally the CPU can be broken down into two subsections, one of which is used to control the timing and sequence of operations in the system, whilst the other executes the required arithmetic and logic operations and handles the data being processed. A memory system is connected to the CPU and is used to store the list of instructions to be executed, known as the *program*, and the data being processed. In most systems a common memory is used to hold both the program instructions and the data but some types of processor use separate memory systems for the data and the instructions. Communication with the outside world is handled by a number of *input* and *output* ports, which allow data to be transferred to and from external devices such as keyboards, display units and printers. The various components of the microcomputer system are tied together by a system of bus lines which are common to all units. This is, of course, a very much simplified description of a microcomputer system and we shall now go on to look at each section in more detail.

BUS SYSTEMS

Data is transferred between the various units of the system over sets of parallel wires known as *buses*. In most systems there are three sets of bus wires, one carrying data, a second carrying memory address information and the third carrying a selection of control signals.

The data bus allows signals representing either data or program instructions to be transferred between the CPU and either the memory or the input–output ports. This bus is always bidirectional and its operation is controlled by the CPU. Read and write control lines from the CPU determine the direction of data flow through the data bus so that when a write operation is performed the signals always flow from the CPU out to memory or I–O ports. A read operation causes signals to flow into the CPU from the memory or I–O port. Many processors, such as the Motorola types, use a single read/write (R/W) control line to control the direction of signal flow on the data bus with one state of the control line indicating a read operation and the other indicating a write operation. Other types, such as the Intel and Zilog processors, have separate read and write control lines. Normally the data bus is set up for read operations as a default condition. There may be several memory or I–O devices which can access the data bus but only one may be allowed to actually drive the bus lines at a time so the bus drive circuits are either tri-state or wired OR type circuits.

The address bus is used to provide an address signal which selects one particular location within the memory for connection to the data bus. The address bus lines are driven by output signals from the CPU. The address bus may also be used to select individual input or output channels where several are connected to the CPU system.

Some processors use separate memories to hold the program instructions and the data. In such a system there may be one data and address bus system for the instruction memory and a separate data and address bus scheme for the data memory. Another variation uses a common address bus with separate data buses for the instructions and data.

The control bus provides a selection of control signals

to and from the CPU which govern the timing and control of data transfers on the other bus lines. Among these may be signals for halting the operation of the CPU and perhaps disconnecting it from the bus system. This facility is important when it is desired to have another device take over control of the bus system. Typical applications might be in multiprocessor systems where a common bus is to be shared between two or more CPUs only one of which may control the bus at any time. The usual scheme is for the external device to send a bus request signal to the CPU when it wants to take over the bus system. On receipt of the bus request the CPU completes its current instruction then disconnects itself from the bus and outputs a bus grant signal to indicate that the bus is free. When the bus request signal is removed the CPU again resumes control of the bus system.

In a large system the output drivers of the CPU chip and other devices may not be capable of driving all of the loads on the bus. In such cases bus drivers, or bus transceivers are used to drive the common bus system.

CPU CONTROL UNIT

Apart from the system timing and control logic the control section of the CPU contains a register called the *program counter* (PC), an address register, an instruction register, a stack or stack pointer register and some interrupt logic.

The instructions which tell the CPU what to do consist simply of a sequence of numbers which are held in the memory. Each instruction usually consists of an *opcode* which defines the type of operation to be carried out and one or more operands which define the data to be used and what is to be done with any results.

The program counter register holds the memory address where the opcode for the next instruction is held. When a program is initiated the program counter is loaded with the address for the first program instruction. As each instruction is executed the program counter is automatically updated to point to the address of the next instruction to be executed. Usually instructions are executed in sequence but occasionally the program sequence may jump to some new point in memory specified by the instruction that has just been executed.

The first stage in executing an instruction is the *fetch cycle* when the opcode for the instruction is read in from the memory and placed in the instruction register. During the next phase of execution the opcode is decoded and the control logic within the CPU is set up to perform the desired operation. During the decoding phase the CPU will determine whether it needs to read in any operand data from the memory. If data is required the next phase will be to read in the operand from memory. The final phase is the actual execution of the instruction.

The timing of the execution sequence varies according to the type of processor involved. In a processor such as the 6800 a simple two phase clock program is used and all data transfers to and from the memory are made during the second half of each clock cycle when the phase 2 clock signal is high. A simple instruction such as Clear Accumulator A (CLRA) consists of an opcode only and executes in two clock cycles. The first cycle is used to read the opcode from memory. During the first half of the next clock cycle the decoding is carried out and then the instruction is executed during the second half of that clock cycle. If the accumulator is to be loaded with immediate data, stored in the memory location following the opcode, the instruction takes up three cycles. The opcode is fetched on the first cycle then the address register is incremented and the data is read in from memory during the second cycle. If the data is to be loaded from an absolute address the number of cycles required goes up to four. Again the opcode is fetched during the first cycle and the next two cycles are used to read in the 16 bit address operand from the two memory locations following the opcode. This address operand is then transferred to the address register and during the fourth cycle the data is read in from the appropriate address in memory. In other types of processor such as the Z80 and 8086 each instruction cycle consists of three or four clock cycles. The clock in such processors usually runs at about four times the speed of the simple 6800 type clock so that actual instruction execution times are similar for both types of processor.

One problem with simple processors is that each instruction must complete its execution sequence before the next instruction is started. To speed up instruction execution most of the modern 16- and 32-bit processors use a pipeline system for instructions and data. In such a system several instructions may be in progress at the same time. Whilst one instruction is being executed the opcode for the next instruction can be decoded and the opcode for the following instruction can be fetched from memory and placed in the pipeline queue. With this overlapped processing of instructions the throughput of the processor can be increased by a factor of two or three over that of a system where each instruction is processed completely before the next starts.

One problem with fast 16- and 32-bit processors is that the speed of access to the main memory system is often much slower than the rate at which the CPU can process instructions. To overcome this situation many systems use a high speed cache memory, built from fast static RAM devices, to hold a block of recently executed instructions. In many program applications the process operates in relatively short loops where a sequence of instructions is repeated a number of times. If this sequence is held in high speed cache memory then the program execution will not be slowed by repeated accesses to the slow main memory. When the next requested instruction is not in the cache memory a cache miss occurs and the main memory is again accessed. Some systems use a similar cache system for data where recently accessed data is held in a cache memory and here again execution can be speeded up where say a table of data values is being processed.

CPU EXECUTION UNIT

The central part of the execution unit of a microprocessor is the arithmetic and logic unit or *ALU* which performs all of the arithmetic and logic functions specified for the execution of an instruction. In most processors the ALU works in conjunction with a special register called the *accumulator*.

The ALU has two data inputs and one data output and these may be either 4, 8, 16 or 32 bits wide according

to the type of processor. The accumulator register has the same number of bits as the ALU. Under normal conditions the accumulator provides one input to the ALU whilst the other input is fed either from another CPU register or from the external data bus. Output data from the ALU is written back into the accumulator.

In some processor types the external data bus has fewer bits than the ALU. For example an 8088 has a 16-bit ALU but only an 8-bit data bus whilst the 68000 has a 32-bit ALU with a 16-bit data bus. In such cases if data is taken from memory to the ALU it is read in as two bytes or words and then assembled in an internal register before being applied to the ALU input. Processors such as the 6809 have two separate 8-bit accumulators which may be used independently or linked together to handle 16-bit numbers.

Apart from the ALU the execution unit will contain a status register which gives information about the result produced by an instruction. This normally contains a series of flag bits which indicate if the result was zero, minus or has produced a carry or an overflow condition. These flags can be used to determine the future flow of program execution according to the state of one or more of the flag bits.

All processors contain some registers which can be used for storing data and temporary results. An important advantage of holding data within on chip registers is that access to register data is always much faster than access to data in the main memory so that program execution can be speeded up by using the internal registers. Most of the newer 16- and 32-bit processors such as the 68000 series have a bank of general purpose registers each of which may be used in the same way as a dedicated accumulator register.

Most processors also contain data pointer registers which can be used to hold memory address information. In this case the pointer register is specified as an operand instead of using an actual memory address. The register can usually be specified in the instruction opcode so that there is no need to read in an operand from memory and thus execution is speeded up. This is particularly useful where a table of data is being accessed and items in the table are stored in successive memory addresses. To move through the table the pointer register is incremented after each instruction to point to the next memory address. In such a scheme the pointer register is usually referred to as the index register and in many types of processor the incrementing or decrementing of this register can be included as part of the action of the instruction.

Typical arithmetic and logic functions provided by the ALU and its associated accumulator are ADD, SUBTRACT, AND, OR and EXCLUSIVE OR. In addition it is usually possible to set the data to zero or to set all the bits to the 1 state. Data held in registers can also be incremented or decremented. It is also possible to shift the data pattern left or right in a register or to rotate the data pattern so that bits spilling from one end of the register are re-inserted at the other end to produce a loop of data bits. In the 6800 series processors memory locations can be used as registers and data in them can be incremented, decremented, shifted and rotated directly.

All of the 16- and 32-bit processors and some 8-bit types provide simple multiply and divide instructions which can handle either signed or unsigned data. These functions generally use an internally stored sequence of operations and may take up many instruction cycles. A few more sophisticated types such as the Am29000 use parallel logic arrays within the ALU to perform hardware multiplication or division and give high execution speed.

Most processors provide only integer arithmetic which may use either pure binary or BCD number systems. For many applications floating point arithmetic may be required in order to deal with large numbers and fractional quantities. For the standard processors this is usually accomplished by using software routines which tend to be relatively slow in execution. An alternative approach is to add a dedicated floating point coprocessor which is designed specifically to carry out floating point operations. The main CPU now passes data and instructions to the coprocessor and then reads back the results and this provides a much higher execution rate for programs requiring floating point calculations. A few of the more advanced processors such as the 80486 and some of the RISC type processors have the floating point execution unit built into the main processor chip.

A very important facility in all microprocessors is the ability to test the results produced by executing an instruction and then to take alternative courses of action according to the results obtained. The tests usually set or reset individual bits in a special register called the status register which may also be referred to as the condition code register. There are four flag bits Z, M, C and V which are common to all processors. The Z bit indicates that the result was zero and the M bit indicates a minus sign or a negative result. The C bit shows that a carry has been generated by an arithmetic or logic operation. The fourth status bit, usually labelled V, indicates an overflow condition where the result is outside the range of numbers that can be correctly represented in the accumulator register. Sometimes a half carry bit is provided which is used when handling numbers in the BCD format. Other bits in the status register may be used to indicate whether interrupts are enabled or disabled and to indicate whether the processor is operating in system or user mode.

The simplest form of conditional instruction is a skip operation. The SKIP instruction examines the status bits and compares them with a specified set of conditions. Thus SKIPZ would check the state of the zero (Z) bit. If the Z bit is set, indicating that the result of the last operation was zero, the program execution jumps over the next instruction after SKIPZ. If the result was not zero then the instruction after SKIPZ is executed. This allows the possibility of two alternative courses of action depending on whether the result of the test was true or false.

In most processors a conditional instruction called a branch is used in conjunction with the status register bits to control the flow of the program execution. The branch instruction tests for a particular state of one or more of the status bits. When the state is true the program execution branches to a new point specified by the operand of the branch instruction. If the specified condition is not met the program continues with execution of the next instruction in the normal way. Usually the operand of the Branch instruction is added to the value in the program counter to calculate the new address from which the program will continue to execute. Some processors have conditional jump instructions which perform in much the same way as a branch

except that the operand specifies the actual address to which the program execution must jump.

SUBROUTINES AND STACKS

In any program there are some sequences of instructions which are frequently repeated as the program executes. Typical examples might be the routine to read a character in from a keyboard or some arithmetic routines such as number conversions or iterative calculations. Whilst these instruction sequences could simply be repeated at appropriate points in the program a more convenient technique is to make use of a subroutine. In this technique the sequence of instructions is stored at a separate point in memory from the main program. The subroutine sequence is often placed immediately after the end of the main program code. To execute the subroutine sequence a special instruction such as CALL or JSR (jump to subroutine) is inserted in the main program sequence at the point where the subroutine is to be executed. This instruction simply tells the CPU that instead of executing the next instruction in the main program it should jump to the first instruction of the subroutine sequence. Before making this jump however the contents of the program counter are saved by the CPU. The short set of subroutine instructions is then executed. At the end of the subroutine sequence another special instruction called RET (return) or RTS (return from subroutine) is executed. This instruction restores the saved contents of the program counter so that the next instruction in the main program sequence will be the one to be executed next.

In simple processors a single save register within the CPU is used to hold the old program counter value whilst the subroutine executes. If a call were made to another subroutine from inside the first subroutine a new value would be written into the save register. At the end of the second subroutine the program would return correctly to the first subroutine but the original PC value in the main program would have been lost and the program would crash because at the end of the first subroutine it would not know where to go next. To overcome this some processors use two or three registers which form a last in first out or LIFO memory which is more generally called a *stack*. This arrangement works in a similar fashion to building a pile of cards. The first PC value is stored in the lowest register in the stack which acts as if it were a single card on a table. When another subroutine is called the PC is placed into the next higher register which is like placing a new card on top of the first. As each new subroutine is started a new higher position register is used and a new card is added to the pile. As a subroutine ends it reads the PC value from the top register in the stack and the next register down becomes the new top of the stack. This is equivalent to removing a card from the pile.

In most processor systems the stack is created in the memory rather than using dedicated save registers. This is achieved by using a stack pointer register which holds the address of the top of the stack which is usually the empty location to which the next data value will be saved. In most systems the stack is arranged to build downwards in memory. Thus after a data word has been written to the stack the stack pointer contents are decrementesd to point to the next free location in the memory. When a word is taken off the stack the pointer is incremented to pick up the last word written to the stack then the word is read out and its location becomes the new top of the stack.

INTERRUPTS

When communicating with the outside world there will be occasions where the processor is ready to transfer data but the external device is not or vice versa. One solution to this problem is to place the processor in a program loop where it repeatedly checks the state of the external device to see if it is ready to send or receive data. As an example if the processor is outputting data to a printer it has to wait for the printer to complete its printing operation before a new data character can be output. A typical printer might operate at perhaps 100 characters per second but during the 0.01 second period for printing a character the processor could have executed some 5000 or more instructions. In the case of keyboard input the program would have to be written so that it checked the keyboard at regular intervals to see if a key had been pressed. These approaches which use a regular testing loop are usually referred to as polling routines. In a real time controller application the processor will need to respond immediately to a number of inputs and if the polling techniques is used most of the available processing time could be spent in checking the status of external devices.

An alternative and much more efficient method off dealing with external devices is to make use of an interrupt system. In an interrupt scheme when the external device is ready to transfer data it sends a signal to a special interrupt request (IRQ) input on the processor chip. When the IRQ input occurs the processor completes its currently executing instruction and then branches to an interrupt service routine which deals with the data transfer to or from the external device. In some ways this is like having a subroutine call which is initiated by an external hardware signal. Before branching off to the interrupt service routine the CPU will automatically save the program counter and the status register. Some processors such as the 6800 save all of the internal CPU registers to the stack when an interrupt occurs. At the end of interrupt service routine there is a return from interrupt (RTI) instruction which causes the saved CPU register contents to be restored so that the program resumes execution from the point where it was interrupted. In most of the 16- and 32-bit processors interrupts are referred to as exceptions but the action is the same.

The simple form of interrupt is usually referred to as a non-maskable interrupt (NMI) and the interrupt service routine is automatically invoked whenever an input is applied to the NMI input line. The alternative type of interrupt scheme is the masked interrupt which is usually triggered by an input to an interrupt request (IRQ) input. When this type of interrupt occurs it sets a mask bit in the status register which causes the IRQ input to be disabled so that any further input signals are ignored whilst the interrupt is being serviced. At the end of the interrupt routine the mask bit is reset and the

IRQ input becomes active again. The mask bit can also be controlled by the program so that the interrupt input can be enabled or disabled as desired.

Another type of interrupt is the software interrupt (SWI) or TRAP operation in which an instruction in the program invokes a branch to an interrupt service routine. Sometimes TRAP operations may be invoked by an error condition such as a divide by zero error. The main difference between a subroutine and a trap is that the subroutine call specifies the address to which the program must jump whereas the trap does not.

Most processors use a vectored interrupt or exception scheme. When the interrupt or exception occurs the address for the start of the interrupt routine is read from a vector table in memory. The position of this address in the table is determined by the type of interrupt that has occurred. As an example in a 6800 processor the vector table is located at the top of the memory map in locations $FFF8–$FFFF. When an NMI input occurs the branch address for the interrupt service routine is take from locations $FFFC and $FFFD. If an IRQ input triggered the interrupt the address comes from locations $FFF8 and $FFF9 whilst an SWI software interrupt causes a branch to the address held in locations $FFFA and $FFFB. In the more complex processors such as the 68000 or 8086 there may be a table of perhaps 256 vector addresses each of which may be assigned to a particular type of interrupt or exception. In some systems the vector address is provided by the external device which caused the interrupt. In this case when the CPU has detected the interrupt it completes its current instruction, outputs an interrupt acknowledge signal and then expects to receive the vector address from the external device via the data bus. Once this address has been read in it is transferred to the program counter and the program branches to the routine at that address in memory.

In most processor systems the different types of interrupt or exception may be assigned a priority level. In such a scheme when an interrupt is being serviced it may itself be interrupted if a higher priority interrupt occurs but interrupts of the same or lower priority levels are masked until the current interrupt service routine has been completed. The highest priority is usually assigned to RESET which triggers an initialisation routine and is used when the system is powered up. The NMI type interrupt always has a higher priority than the normal masked interrupts.

MEMORY

The microcomputer memory is used to hold the list of program instructions and any data used or produced by the program. The memory itself consists of a vast array of individual cells each of which can hold one bit of data. The array is normally arranged so that a complete byte or word of data can be read from or written to the memory in parallel. The main memory is usually based on dynamic memory devices where the data is stored as a charge within the memory cell. The main problem with this type of memory is that due to leakage paths within the cell the data is held reliably for only a few milliseconds. To maintain the data storage a process of *refreshing* is used where the data is read from the memory and rewritten at intervals of about 2 milliseconds. An alternative type of memory device uses a flip-flop type circuit for each memory cell which once set will retain its data state indefinitely unless new data is written in. This type of memory is referred to as a *static* memory and will generally operate faster than a similar dynamic type. The main disadvantage of the static memory is that because the circuit is more complex static types are smaller in storage capacity than similar dynamic types. Both types are usually addressed so that any word in the memory array can be selected directly and are referred to as a random access memory or RAM.

In a general purpose microcomputer system the program and data are usually stored in read/write RAM but some initialisation routines or an operating system are usually in a read only memory (ROM) so that on power up the processor is ready to read in a program to be executed. In this type of system the programs are usually held on some form of external media such as floppy disks and the system ROM will contain the system program routines needed to load in a new program for execution. In single chip microcontrollers the program is usually stored in on chip ROM or EPROM so that on power up the system automatically starts to execute its program. Some data such as constants may also be held in the ROM whilst on chip RAM is used for temporary data storage and as working space for the program.

Most 8-bit processors have a 16-bit address bus which allows them to access up to 64k bytes of memory. The older 16-bit processors such as the 8086 and 68000 have a 20- or 24-bit wide address bus giving direct access to 1 or 16 megabytes of memory. Later types of 16- and 32-bit processors use a full 32-bit wide address which can directly access up to 4 gigabytes of memory. In order to reduce the number of pins required on the device package some processors multiplex the data and address bus lines on to the same set of pins. For these types there is usually an address latch output which indicates when an address is set up on the bus. The address information is then transferred to an external latch which drives the memory address system whilst the data transfer is performed.

Many of the newer processors are also designed to operate using a virtual memory scheme similar to that used on minicomputers and mainframe machines. This type of memory scheme is generally used when the processor is handling a multi-tasking or multi-user environment where several different programs may be in progress at the same time. In fact only one program is actually being executed at any instant in time but other programs are being held in memory. When the executing program is held up whilst waiting for an external device such as a printer it is temporarily suspended and another program starts executing. When the first program is ready to start execution again the new program is suspended and the first program resumes execution. The programs held in memory are normally assigned levels of priority and the CPU executes the highest priority program that is not being held up for input or output. The switching of programs for execution is carried out by a special operating system program called a scheduler.

One problem with multi-tasking is that the physical memory attached to the CPU has to be shared between the various programs or tasks that are scheduled to be

processed. In a virtual memory system each task may be allowed access to a memory space which can be equal to the size of the maximum physical memory space of the computer. The memory space referenced by the program is called a *virtual* memory. Since there is only a limited amount of physical memory the scheduler allocates blocks or pages of this memory to each of the tasks being processed. Thus for each task only a small part of the complete program is loaded into memory at a time. When this section of program has completed execution or the instructions call for a jump to a new section of program then a new chunk of program is loaded in from the disk memory. In the memory management unit the virtual program memory address is translated to an equivalent address in the block of physical memory assigned to that program.

A multi-user system operates in a similar fashion to multi-tasking with each user being treated as a separate task. The users are often placed at the same priority level and the use of the processor is then shared by allocating each user a fixed time period and then cycling through the user programs in sequence. If the user program is able to use its time slot it will execute but if it is not ready to execute then control passes to the next user in sequence. In these shared task schemes all of the resources of the system are allocated by the scheduler including any external devices such as a printer or disk memory.

INPUT–OUTPUT

To be of any use the microprocessor must be able to communicate with the outside world. In a typical system there might be a keyboard or keypad which is used for input and a video display or printer for output. Digital data may be transferred between the CPU and these input and output devices via channels called ports.

A typical output port consists of a data register whose inputs are connected to the data bus when data is to be output. After the data transfer the register retains the data pattern and provides a constant signal on a set of output lines. An input port may also contain a latch register and the output of this is briefly connected to the data bus to allow data to be transferred into the CPU.

In single chip microcontrollers the input and output port registers are part of the CPU itself and instructions are included for carrying out data transfers to or from the input and output ports. In a general purpose processor system the input and output port registers are often treated as if they were locations in the memory and a part of the memory map address space is set aside for them. Each port can then be selected for data transfer by using the appropriate memory address in the load or store instruction. Some processors such as the Intel types have special IN and OUT instructions for data transfer to I/O ports and assign a separate address map for input–output operations. In this case the state of an M/IO control line indicates whether the data transfer is to memory or an I/O channel and the lower 8 bits of the address bus are then used to address the input-output ports if several are connected.

Apart from their data lines the input and output channels may also have additional lines which are used for handshaking. These usually consist of a request line which indicates that a data transfer is required and a status line which indicates that the data transfer has been performed. In the case of a printer system the processor would test the status signal from the printer to see if it was busy printing a character. If the printer were not busy the data would be placed on the port data lines and a strobe pulse would be output on the request line to tell the printer to accept a new data word and print the corresponding character. As soon as the printer accepts the data it would set its status line to busy until the character had been printed. These handshake lines are particularly important where the processor and the external device are operating at different speeds.

WORD LENGTH

Microprocessors work with binary data consisting of groups of binary digits or bits which are called data words. In a typical microprocessor system the words may be made up from 4, 8, 16 or 32 bits according to the type of processor being used.

Early microprocessors and many modern low cost microcontrollers work with 4-bit data words which are sometimes referred to as *nibbles*. In these processors the data bus and the ALU are all 4 bits wide. A 4-bit word in which each bit can be either on or off can have 16 possible combinations and could be used to represent the numbers from 0 to 15. Larger numbers are processed by dealing with the data in 4-bit segments. The 4-bit microcontrollers are often used to process data in the binary coded decimal format where each decimal digit of the number is coded as a 4-bit word with a value from 0 to 9.

Many of the popular general purpose microprocessors such as the 6502, 6809 and Z80 use an 8-bit data word which is generally referred to as a *byte*. In these processors the ALU and data bus are 8 bits wide and the numerical value represented by a single data word can be from 0 to 255 or from −128 to +127 if a signed number format is used. The byte is also convenient for representing alphanumeric character codes where up to 256 different characters can be defined. Once again large numbers can be represented by using two or more data bytes.

More recently general purpose processors such as the Intel 80x86 and Motorola 68xxx series use 16-bit or 32-bit data bus systems. In most cases the width of the data bus is used to define the type of processor but there are some anomalies in this definition. In the 68000 processor the internal registers and data bus are 32 bits wide but the external data bus is only 16 bits wide so the processor is sometimes referred to as a 16/32-bit processor. The 8088 is another case where the internal system is the same as that of the 16-bit 8086 but the external data bus is only 8 bits wide.

In the 16-bit processors a data word is normally 16 bits wide and a 32-bit data value is usually referred to as a long word or double word. For 32-bit processors it is usual to define a 16-bit value as a half word and use the name word for a 32-bit value.

TYPES OF DEVICE

Microprocessor based devices are usually available in three forms which are dedicated microcontrollers or

microcomputers, general purpose microprocessors and bit slice devices.

The first type of device is the single chip microcomputer which is often called a microcontroller. This type of chip contains the CPU, some ROM or EPROM to contain the program and some RAM for use as working storage. In addition these chips usually contain several input–output ports for both parallel and serial input and output. There may also be on chip counter-timers and perhaps an analogue-to-digital converter to handle analogue inputs. This type of device is normally used for dedicated applications such as controllers in domestic appliances, video recorders or industrial equipment.

The general purpose microprocessor usually contains only the basic CPU functions including the ALU, working registers and general control logic. Address and data bus systems are used to communicate either with an external memory which holds the program and its data or with a series of input–output ports for communication with other devices.

Among the popular 8-bit general purpose microprocessors are the 6502 and Z80 which are widely used in home computer systems and the 6809 which is popular for industrial applications. Among the more powerful 16-bit microprocessor devices are the 8086 and 68000 which are widely used in personal computer systems for home and business use. The more powerful personal computers and professional workstations use 32-bit processors such as the 80386 or 68030.

One trend in the general purpose microprocessors has been that more and more complex instructions have been built into these devices as their power increased. This type of processor has generally come to be referred to as a complex instruction set computer or CISC type. After some analysis of the actual activity of such a processor when executing a typical program it was found that the frequently used instructions represented only a part of the available instruction set and that in general these tended to be relatively simple operations. The result was that a different approach was tried in the design of the processor. In this approach the basic widely used operations were implemented and most operations were performed on data held in on chip registers. The result was that instructions could be executed quickly and although there were more instructions making up a program the actual execution speed could still be higher than that of a CISC type processor. This new style of processor is referred to as a reduced instruction set computer or RISC type and has become popular particularly for applications which require large amounts of repetitive processing operations such as image processing and complex data analysis tasks.

The third type of processor device is the bit slice processor where each function of the processor is built up from small segments known as slices. Thus the ALU might be assembled from a number of 4-bit wide ALU elements each fabricated on a separate chip. In the same way the control section which is usually referred to as a microsequencer is also built up from 4- or 8-bit wide segments. The early types of bit slice processors were fabricated using high speed bipolar logic and were used to build very fast processors. More recently this type of processor device has been fabricated using CMOS techniques and has retained high execution speed with very low power consumption.

FABRICATION TECHNOLOGY

The earliest types of microprocessor device were based on the technology of chips designed for electronic pocket calculators and often used PMOS (p channel metal oxide semiconductor) type circuits which were based around p channel field effect transistor elements. This was largely because this type of integrated circuit was easier to fabricate than circuits based on n channels fets. As technology improved most microprocessors were fabricated using NMOS (n channel metal oxide semiconductor) techniques which provided higher operating speed and allowed the circuits to operate from a single 5 volt power supply. Many of the popular 8- and 16-bit types used today are NMOS devices.

An alternative form of fabrication which became popular in the late 1970s is complementary metal oxide semiconductor or CMOS which combines the use of p and n channel fets on the same chip. The major advantage of the CMOS type of device is that its power requirements are very much lower than a similar NMOS type and thus processors fabricated using CMOS were ideal for low power applications such as portable and battery powered equipment. Early CMOS types tended to be slower than similar NMOS types but improvements in CMOS fabrication technology has resulted in modern CMOS processors which are just as fast as their NMOS counterparts whilst providing lower power requirements.

An important factor to be considered when using NMOS and CMOS devices is that they have extremely high impedance inputs and can be prone to damage from static electricity charges. Although all modern devices have built in protection diodes to help prevent the build up of static charges on the device inputs it is still a wise precaution to store these devices with the pins shorted together via metal foil or conductive plastic and to take precautions against the build up of static electric charges when handling such devices.

CHOOSING A MICROPROCESSOR

Unfortunately for the system designer there is no convenient magic formula by which the optimum microprocessor or microcomputer device can be selected for a particular application. It is of course fairly easy to choose one or more processors which may be technically suited for the project but generally the final choice will be dictated by software and economic considerations.

Basically the process of choosing a suitable microprocessor can be broken down into the following stages:

(1) Define exactly what the system is going to do.

(2) Decide whether the system should be based on the use of a general purpose microprocessor or a single chip microcontroller.

(3) Choose the most suitable word length for the application.

(4) Consider the hardware factors such as speed, power requirements and the availability of existing hardware modules.

(5) Consider the software design with particular regard to in-house expertise and available in-house development aids.

(6) Examine any economic factors.

At each of these stages it should be possible to eliminate a number of the available microprocessor types until there are perhaps two or three possible contenders from which a final choice can be made.

The first stage may seem very obvious and yet it is surprising how many system designers will progress to detailed design before they have defined exactly what is required. At this stage the system specification can be divided into two broad areas. First there are the requirements that are absolutely essential and secondly there are features which, whilst not essential, may be desirable since they will produce a more versatile or more attractive product. These features may be arranged in a list with some sort of priority or value rating given to each item. The secondary system requirements may become useful later in the selection process where a choice has to be made between two more or less equally suitable devices.

Once the system requirements have been decided it is important to ask the question 'Is a microprocessor needed at all?' Consider what would be involved in meeting the system requirements using conventional discrete logic devices, programmed logic arrays or off-the-shelf dedicated circuits. In most cases it is likely that these approaches will be impractical and a microprocessor based system is then inevitable. Where these other approaches are possible alternatives they should be considered along with the microprocessor based solutions. It would be ridiculous to use a microprocessor based system when a simple logic system could provide a cheaper or simpler solution.

The choice between a bit slice system, an embedded single chip microcontroller or a general purpose microprocessor can usually be determined by technical considerations. The bit slice approach may be attractive for very high speed or rather specialised applications but generally the choice will be between a single chip microcontroller or a general purpose microprocessor.

The single chip microcontroller is most suitable for applications where space is limited or for equipment which is likely to be produced in large quantities. This type of device is widely used in consumer equipment such as domestic appliances, video recorders, microwave ovens, TV receivers and automobile systems. The embedded controller is also attractive in various types of industrial equipment such as digital servo systems and test instruments where a fixed program can be used.

Most of the standard versions of microcontroller chip use a mask programmed on chip ROM to carry the program instructions. The initial cost of producing the mask is likely to be quite high and normally a production run of some 5000 or more units will be required in order to justify this initial cost. For a project where mass production can be used the cost per unit for the single chip embedded controller approach will be much less than the cost of using a general purpose microprocessor based system. Another factor to be considered here is that the single chip controller will have fewer lead connections and is thus likely to be more reliable than a multichip approach. The size of circuit boards and the cost of assembly is also likely to be lower when a single chip controller is used.

For prototype and small batch production the mask programmed single chip device becomes uneconomical. Most of the manufacturers of such chips do however produce alternative versions which use an on chip PROM or EPROM instead of a mask programmed ROM. This type of device can therefore be programmed in the field and is ideal when only a small number of units are to be produced. Typical examples of such devices are the 8748 and 8751 series from Intel and the 68701 or 68705 devices from Motorola. The EPROM versions are ideal for prototype development since the EPROM can be erased by using an ultraviolet light source and can then be reprogrammed with a modified version of the software program. This process may be repeated a number of times until the system performs correctly. The versions which have a PROM on the chip are often referred to as 'one time programmable' or OTP devices since they cannot be erased. This type of device is suited to production runs involving perhaps a few tens or hundreds of units.

For applications where a wide range of options are required or where the programming must be flexible to cope with a range of different tasks the general purpose microprocessor approach is ideal. Typical of such applications are general purpose computer systems such as personal computers, point of sale systems, production control systems and graphics or CAD workstations. Other applications where this type of device is appropriate might be robotics, numerically controlled machine tools and image processing. This type of application can often be handled by one of the popular processors such as the Z80, 6809, 8086 or the more advanced 80386 or 68000 series types. For high speed applications or those where there is very high level of mathematical computation, such as in image processing, one of the newer RISC type processors is likely to be attractive.

The word length can often be determined from the basic technical specifications for the system. In a simple appliance controller which is basically replacing a discrete logic sequencer the 4-bit processor is ideally suited. Inputs can be fed in as BCD digits from a keypad and outputs are usually made to simple digital indicators such as LED, LCD or VF type numeric displays. The 8-bit word length becomes attractive where text information is to be handled and is suitable for more complex controllers particularly those which have analogue input requirements and where some degree of computation is required as well as simple logic sequence control. The 8-bit types also have the capability of accessing a large external memory of up to 64 kbytes and are useful for applications involving either large programs or large amounts of data.

For applications involving general purpose computing tasks the 16- or 32-bit word length are usually better than 8-bit types since they usually execute programs at a much higher speed and are capable of handling very large memory systems. If the system is to perform multitask operations or have a mutli-user capability then the modern 32-bit microprocessors such as the 68030 or 80386 become essential.

The choice of fabrication technology generally comes down to a decision of whether to use conventional NMOS type devices which have fairly high power requirements or to use CMOS versions which operate at much lower power. In the early days CMOS devices tended to be slower in operation than NMOS types but improvements in CMOS technology have made modern CMOS processors just as fast as their NMOS equivalents. For battery operated or portable equipment CMOS types are the obvious choice but for other applications where power demand is not a consideration the