

UNIV. OF MD COLLEGE PARK



3 1430 04785887 5

Computer Busses

DESIGN AND APPLICATION

W. BUCHANAN

SIGN AND APPLICATION

DESIGN AND APPLICATION

COMPUTER BUSES
COMPUTER BUSES

DESIGN AND APPLICATION

COMPUTER
BUSES

Computer Busses

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND

DEC 11 2000

ENGINEERING LIBRARY
COLLEGE PARK, MD

Computer Busses

William Buchanan, BSc (Hons), CEng, PhD
Napier University, Edinburgh, UK



A member of the Hodder Headline Group
LONDON



Co-published in North America by CRC Press
Boca Raton • New York • Washington, D.C.

First published in Great Britain in 2000 by
Arnold, a member of the Hodder Headline Group,
338 Euston Road, London NW1 3BH

<http://www.arnoldpublishers.com>

Co-published in North America by CRC Press LLC,
2000 N.W. Corporate Blvd.,
Boca Raton,
Florida 33431
USA

© 2000 William Buchanan

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronically or mechanically, including photocopying, recording or any information storage or retrieval system, without either prior permission in writing from the publisher or a licence permitting restricted copying. In the United Kingdom such licences are issued by the Copyright Licensing Agency: 90 Tottenham Court Road, London W1P 0LP.

Whilst the advice and information in this book are believed to be true and accurate at the date of going to press, neither the author nor the publisher can accept any legal responsibility or liability for any errors or omissions that may be made.

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

Library of Congress Cataloging-in-Publication Data

A catalog record for this book is available from the Library of Congress

ISBN 0 340 74076 0

ISBN 0 8493 0825 9 (CRC Press)

1 2 3 4 5 6 7 8 9 10

Printed and bound in Great Britain by J W Arrowsmith Ltd, Bristol

What do you think about this book? Or any other Arnold title?
Please send your comments to feedback.arnold@hodder.co.uk



Table of Contents

	Preface	<i>xiii</i>
1	INTRODUCTION	1
	1.1 Pre-PC Development	1
	1.2 8008/8080/8085	6
	1.3 8086/8088	13
	1.4 80186/80188	19
	1.5 80286	20
	1.6 Post-PC development	21
	1.7 Exercises	36
	1.8 Notes from the author	40
	1.9 DEC	45
	1.10 Open .v. closed systems	47
	1.11 RIP, Sinclair Research	48
	1.12 How to miss a market	48
2	BUSSES, INTERRUPTS AND PC SYSTEMS	49
	2.1 Busses	49
	2.2 Interrupts	61
	2.3 Interfacing	69
	2.4 PC Systems	76
	2.8 Practical PC system	77
	2.5 Exercises	79
	2.6 Notes from the author	82
3	INTERFACING STANDARDS	85
	3.1 Introduction	85
	3.2 PC bus	85
	3.3 ISA bus	87
	3.4 Other legacy busses	91
	3.5 Comparison of different types	92
	3.6 Exercises	93
	3.7 Summary of interface bus types	95
	3.8 The fall of the MCA bus	97
	3.9 Notes from the author	98
4	PCI BUS	103
	4.1 Introduction	103
	4.2 PCI operation	106
	4.3 Bus arbitration	109
	4.4 Other PCI pins	110
	4.5 Configuration address space	110

4.6	I/O addressing	112
4.7	Exercises	116
4.8	Example manufacturer and plug-and-play IDs	118
4.9	Notes from the author	119
5	MOTHERBOARD DESIGN	121
5.1	Introduction	121
5.2	TX motherboard	132
5.3	Exercises	136
5.4	Notes from the author	137
6	IDE AND MASS STORAGE	139
6.1	Introduction	139
6.2	Tracks and sectors	139
6.3	Floppy disks	140
6.4	Fixed disks	141
6.5	Drive specifications	142
6.6	Hard disk and CD-ROM interfaces	142
6.7	IDE interface	143
6.8	IDE communication	144
6.9	Optical storage	150
6.10	Magnetic tape	153
6.11	Exercises	155
6.12	Notes from the author	156
7	SCSI	157
7.1	Introduction	157
7.2	SCSI types	157
7.3	SCSI interface	159
7.4	SCSI operation	162
7.5	SCSI pointers	164
7.6	Message system description	165
7.7	SCSI commands	167
7.8	Status	169
7.9	Exercises	171
7.10	Notes from the author	172
8	PCMCIA	173
8.1	Introduction	173
8.2	PCMCIA signals	173
8.3	PCMCIA registers	175
8.4	Exercises	179
8.5	Notes from the author	179
9	USB AND FIREWIRE	181
9.1	Introduction	181
9.2	USB	182

9.3	Firewire	186
9.4	Exercises	190
9.5	Notes from the author	190
10	GAMES PORT, KEYBOARD AND MOUSE	191
10.1	Introduction	191
10.2	Games port	191
10.3	Keyboard	195
10.4	Mouse and keyboard interface	198
10.5	Mouse	199
10.6	Exercises	200
10.7	Notes from the author	201
11	AGP	203
11.1	Introduction	203
11.2	PCI and AGP	204
11.3	Bus transactions	205
11.4	Pin description	205
11.5	AGP master configuration	208
11.6	Bus commands	209
11.7	Addressing modes and bus operations	210
11.8	Register description	210
11.9	Exercises	215
11.10	Notes from the author	215
12	FIBRE CHANNEL	217
12.1	Introduction	217
12.2	Comparison	217
12.3	Fibre channel standards	218
12.4	Cables, hubs, adapters and connectors	219
12.5	Storage Devices and storage area networks	221
12.6	Networks	221
12.7	Exercises	222
12.8	Notes from the author	222
13	RS-232	223
13.1	Introduction	223
13.2	Electrical characteristics	223
13.3	Communications between two nodes	228
13.4	Programming RS-232	233
13.5	RS-232 programs	237
13.6	Exercises	241
13.7	Notes from the author	246
14	RS-422, RS-423 AND RS-485	247
14.1	Introduction	247
14.2	RS-485 (ISO 8482)	247

14.3	Line drivers	249
14.4	RS-232/485 converter	250
14.5	Exercises	251
14.6	Note from the author	251
15	MODEMS	253
15.1	Introduction	253
15.2	RS-232 communications	254
15.3	Modem standards	255
15.4	Modem commands	256
15.5	Modem set-ups	258
15.6	Modem indicator	260
15.7	Profile viewing	260
15.8	Test modes	261
15.9	Digital modulation	264
15.10	Typical modems	265
15.11	Fax transmission	267
15.12	Exercises	268
15.13	Notes from the author	269
16	PARALLEL PORT	271
16.1	Introduction	271
16.2	PC connections	271
16.3	Data handshaking	272
16.4	I/O addressing	275
16.5	Interrupt-driven parallel port	279
16.6	Exercises	284
16.7	Notes from the author	287
17	ENHANCED PARALLEL PORT	289
17.1	Introduction	289
17.2	Compatibility mode	289
17.3	Nibble mode	290
17.4	Byte mode	293
17.5	EPP	294
17.6	ECP	296
17.7	Exercises	300
17.8	Note from the author	300
18	MODBUS	301
18.1	Modbus protocol	301
18.2	Function codes	307
18.3	Modbus diagnostics	309
18.4	Exercises	311
18.5	Notes from the author	312

19	FIELDBUS	313
	19.1 Introduction	313
	19.2 Fieldbus types	313
	19.3 FOUNDATION Fieldbus	316
	19.4 Exercises	323
	19.5 Notes from the author	323
20	WORLDVIP	325
	20.1 Introduction	325
	20.2 Physical layer	325
	20.3 Data link layer	326
	20.4 Exercises	330
	20.5 Notes from the author	331
21	CAN BUS	333
	21.1 Introduction	333
	21.2 CAN physical	335
	21.3 CAN bus basics	336
	21.4 Message transfer	337
	21.5 Fault confinement	340
	21.6 Bit timing	341
	21.7 CAN open	342
	21.8 Exercises	342
	21.9 Notes from the author	343
22	IEEE-488, VME AND VXI	345
	22.1 Introduction	345
	22.2 IEEE-488 bus	345
	22.3 VME bus	348
	22.4 VXI bus	349
	22.5 Exercises	352
	22.6 Notes from the author	353
23	TCP/IP	355
	23.1 Introduction	355
	23.2 TCP/IP gateways and hosts	356
	23.3 Function of the IP protocol	356
	23.4 Internet datagram	357
	23.5 ICMP	359
	23.6 TCP/IP internets	362
	23.7 Domain name system	366
	23.8 Internet naming structure	367
	23.9 Domain name server	368
	23.10 Bootp protocol	369
	23.11 Example network	371
	23.12 ARP	373
	23.13 IP multicasting	373

	23.14 Exercises	375
	23.15 Notes from the author	377
	23.16 Additional material	378
24	TCP AND UDP	385
	24.1 Introduction	385
	24.2 Transmission control protocol	385
	24.3 UDP	389
	24.4 TCP specification	390
	24.5 TCB parameters	392
	24.6 Connection states	392
	24.7 Opening and closing a connection	395
	24.8 TCP user commands	397
	24.9 WinSock	399
	24.10 Visual Basic socket implementation	408
	24.11 Exercises	414
	24.12 TCP/IP services reference	416
	24.13 Notes from the author	416
25	NETWORKS	419
	25.1 Introduction	419
	25.2 Network topologies	421
	25.3 OSI model	424
	25.4 Routers, bridges and repeaters	426
	25.5 Network cable types	429
	25.6 Exercises	431
	25.7 Notes from the author	432
26	ETHERNET	435
	26.1 Introduction	435
	26.2 IEEE standards	436
	26.3 Ethernet – media access control (MAC) layer	437
	26.4 IEEE 802.2 and Ethernet SNAP	439
	26.5 OSI and the IEEE 802.3 standard	441
	26.6 Ethernet transceivers	442
	26.7 Ethernet types	443
	26.8 Twisted-pair hubs	445
	26.9 100Mbps Ethernet	445
	26.10 Comparison of fast Ethernet other technologies	450
	26.11 Switches and switching hubs	451
	26.12 Network interface card design	453
	26.13 Gigabit Ethernet	457
	26.14 Exercises	462
	26.15 Ethernet crossover connections	464
	26.16 Notes from the author	465
27	RS-232 PROGRAMMING USING VISUAL BASIC	467
	27.1 Introduction	467

27.2	Properties	467
27.3	Events	473
27.4	Example program	474
27.5	Error messages	475
27.6	RS-232 polling	476
27.7	Exercises	477
28	INTERRUPT-DRIVEN RS-232	479
28.1	Interrupt-driven RS-232	479
28.2	DOS-based RS-232 program	479
28.3	Exercises	486
A	PC PROCESSORS	489
A.1	Introduction	489
A.2	8086/88	490
A.3	80386/80486	495
A.4	Pentium/Pentium Pro	501
A.5	Exercises	505
B	VESA VL-LOCAL BUS	509
C	MODEM CODES	511
C.1	AT commands	511
C.2	Result codes	513
C.3	S-registers	514
D	REDUNDANCY CHECKING	519
D.1	Cyclic redundancy check (CRC)	519
D.2	Longitudinal/vertical redundancy checks (LRC/VRC)	523
E	ASCII CHARACTER CODE	525
E.1	Standard ASCII	525
E.2	Extended ASCII code	527
F	QUICK REFERENCE	529
F.1	Notes from the author	531
G	ISDN	533
G.1	Introduction	533
G.2	ISDN channels	534
G.3	ISDN physical layer interfacing	535
G.4	ISDN data link layer	538
G.5	ISDN network layer	541
G.6	Speech sampling	543
G.7	Exercises	544

H	MICROSOFT WINDOWS	547
	H.1 Introduction	547
	H.2 Windows registry	548
	H.3 Device drivers	550
	H.4 Configuration manager	551
	H.5 Virtual machine manager (VMM)	552
	H.6 Multiple file systems	555
	H.7 Core system components	557
	H.8 Multitasking and threading	559
	H.9 Plug-and-play process	561
	H.10 Windows NT architecture	561
	H.11 Windows 95 and Windows 98	564
	H.12 Fundamentals of Operating Systems	565
	H.13 Exercises	567
I	HDLC	569
	I.1 Introduction	569
	I.2 HDLC protocol	570
	I.3 Transparency	574
	I.4 Flow control	574
	I.5 Derivatives of HDLC	576
J	EXAMPLE WINSOCK CODE FOR VISUAL BASIC	
	J.1 My client (myClient.frm)	579
	J.2 My server (myServer.frm)	583
	J.3 Choice form (ChoiceSC.frm)	586
	J.4 Error panel (ErrorPanel.frm)	587
	J.5 Help form (help.frm)	589
	Index	591

2.1 Busses

The part that makes computers operate and allows devices to be easily plugged in is the computer bus, which allows the orderly flow of data between one device and another. The PC, and other computer systems, has an amazing number of different types of interfaces and bus systems, these include the PC bus, ISA bus, PCI bus, CAN bus, AGP bus, games port, parallel port, serial port, and so on.

The main elements of a basic computer system are a central processing unit (or microprocessor), memory, and I/O interfacing circuitry. These connect by means of three main buses: the address bus, the control bus and the data bus. A bus is a collection of common electrical connections grouped by a single name. Figure 2.1 shows a basic system. External devices such as a keyboard, display, disk drives can connect directly onto the data, address and control buses or through the I/O interface circuitry.

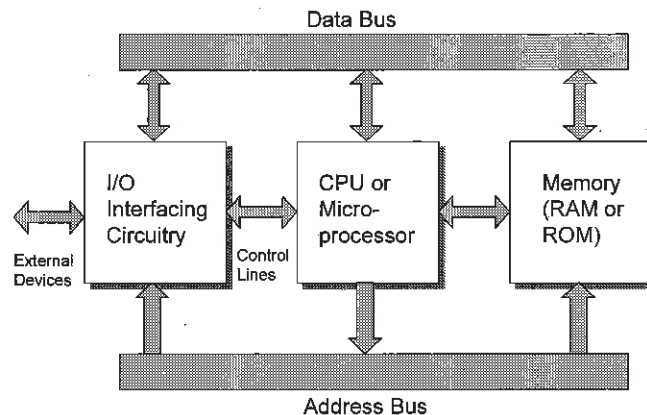


Figure 2.1 Block diagram of a simple computer system

Electronic memory consists of RAM (random access memory) and ROM (read only memory). ROM stores permanent binary information, whereas RAM is a non-permanent memory and loses its contents on a loss of power. Applications of this type of memory include running programs and storing temporary information. RAM is normally made up of either DRAM (Dynamic RAM) or SRAM (Static RAM). DRAM uses a single capacitor and a transistor to store a single bit of data, whereas SRAM uses six transistors, arranged as a flip-flop device, to store a single bit of data. DRAM has the advantage that more memory can be crammed onto a microchip (as only one transistor is required for each bit stored). DRAM, though, has two major disadvantages: it is relatively slow (because of the charging

and discharging of the storage capacitors) and it requires that the complete contents of its memory be refreshed with power many times a second (because the tiny capacitors lose their charge over a short time). This power refresh is thus wasteful of electrical power and leads to heat dissipation.

The microprocessor is the main controller of the computer. It only understands binary information and operates on a series of binary commands known as machine code. It fetches binary instructions from memory, decodes these instructions into a series of simple actions and carries out the actions in a sequence of steps. A system clock synchronises these steps.

To access a location in memory the microprocessor puts the address of the location on the address bus. The contents at this address are then placed on the data bus and the microprocessor reads the data from the data bus. To store data in memory the microprocessor places the data on the data bus. The address of the location in memory is then put on the address bus and data is read from the data bus into the memory address location.

The classification of a microprocessor relates to the maximum number of bits it can process at a time, that is their word length. The evolution has gone from 4-bit, 8-bit, 16-bit, 32-bit and to 64-bit architectures.

2.1.1 Bus specification

The basic specification of a computer can be determined by analysing the performance of the busses within the system. Each bus performs a specific function and is suited to the devices that connect to it. The basic specifications for busses include:

- **Data rate** (in bytes per second or bits per second). This defines the maximum amount of data that can be transferred, at a time. For example, the ISA bus has a maximum data rate of 16MB/s, Gigabit Ethernet has a maximum data rate of 125MB/s, and the local bus which connects a PC processor to local memory can have a data rate of over 800MB/s (64 bits at 100MHz).
- **Maximum number of devices which connect to the bus.** The number of devices which connect to a bus can have a great effect on its performance as they all provide an electrical loading on the bus and the more that connect to the bus, the greater the overhead of bus arbitration will be. Standard SCSI only allows a maximum of seven devices to be connected to the bus, whereas Ethernet can allow thousands of devices to connect to the bus.
- **Bus reliability.** This defines how well the bus copes with any errors which occur on the bus. Some busses, especially in industrial environments, can be susceptible to externally generated noise. A good bus should be able to detect if it has received data which has been corrupted by noise (or was sent incorrectly).
- **Data robustness.** This is the ability of the bus to react to faults within the bus or from the malfunctioning of connected devices. Busses such as the CAN bus can isolate incorrectly operating devices.
- **Electrical/physical robustness.** This is the ability of the bus to cope with electrical faults, especially due to short-circuits and power surges. Problems can also be caused by open circuit electrical connections, although these tend not to cause long term damage to the bus. The physical robustness of a bus is also important, especially in industrial or safety critical situations.
- **Electrical characteristics.** This involves the basic electrical parameters of the bus, such as the range of voltage levels used, electrical current ranges, short-circuit protection sys-

tem, capacitance and impedance of cables, cross-talk (the amount of interference between local signal transmissions), and so on.

- **Ease-of-connection.** This includes the availability of cables and connectors, and how easy it is to add and remove devices from the bus. Some busses allow devices to be added or removed while the bus is in operation (hot pluggable). A good example of a hot-pluggable bus, which is easy to connect to, is the USB.
- **Communications overhead.** This is a measure of the amount of data that is added to the original data, so that it can be sent in a reliable way. Local, fast busses normally have a minimum of overhead, whereas remote, networked busses have a relatively large overhead on the transmitted data.
- **Bus controller topology.** This relates to the method that is used to control the flow of data around the bus. Some busses, such as SCSI, require a dedicated bus controller which is involved in all of the data transfers, whereas the PCI bus can operate with one or more bus controller devices taking control of the bus. Other busses, such as Ethernet, have a distributed topology where any device can take control of the bus.
- **Software interfacing.** This defines how easy it is to interface to the bus with software, especially when using standard interface protocols, such as TCP/IP or MODBUS.
- **Cable and connectors.** This defines the range of cables and connectors that can be used with the bus. There is a wide range of cables available, such as ribbon cables (which are light and are useful inside computer systems), twisted-pair cables (which are easy to connect to and are useful in minimising cross-talk between transmitted signals) and fibre optic cables (which provide a high capacity communications link and minimise cross talk between transmitted signals). For example, Ethernet can use BNC connectors with coaxial cables, RJ-45 connectors with twisted-pair cables and SNA connectors with fibre optic cables.
- **Standardisation of the bus.** Most busses must comply with a given international standard, which allows hardware and software to interconnect in a standard form. There are normally standards for the electrical/mechanical interface, the logical operation of the bus, and its interface to software. For example, the IEEE has defined most of the Ethernet standard (especially IEEE 802.3), and the EIA have defined the RS-232 standard. International standard agencies, such as the IEEE, ISO, ANSI and EIA, provide a more secure standard than a vendor-led standard.
- **Power supply modes.** Some busses allow power saving modes, where devices can power themselves down and be powered up by an event on the bus. This is particularly useful with devices that have a limited power supply, such as being battery supplied.

2.1.2 Bus components

Devices connect to each other within a computer using a bus. The bus can either be an internal bus (such as the IDE bus which connects to hard disks and CD-ROM drives within a PC) or an external bus (such as the USB which can connect to a number of external devices, typically to scanners, joypads and printers). Busses typically have a number of basic components: a data bus, an optional address bus, control lines and handshaking lines, as illustrated in Figure 2.2. Other lines, such as clock rates and power supply lines are not normally displayed when discussing the logical operation of the bus. If there is no address bus, or no control and handshaking lines, then the data bus can be used to provide addressing, control and handshaking. This is typical in serial communications, and helps to reduce the number of connections in the bus, although will generally slow down the communications.

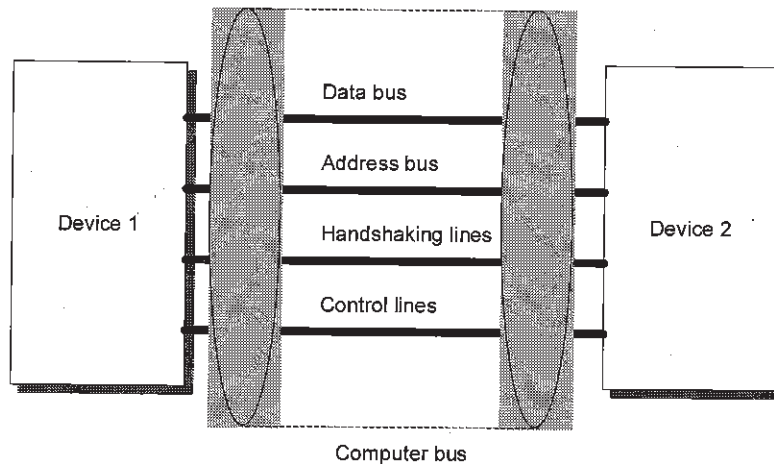


Figure 2.2 Model of a computer bus

Data bus

The data bus is responsible for passing data from one device and another. This data is either passed in a serial manner (one bit at a time) or in parallel (several bits at a time). In a parallel data bus, the bits are normally passed in a multiple of eight bits at a time. Typical parallel data busses are 8 bits, 16 bits, 32 bits, 64 bits or 128 bits wide.

The bus size defines the maximum size of the bus, but the bus can be used to transmit any number of bits which is less than the maximum size. For example, a 32-bit bus can be used to transmit eight bits, 16 bits or 32 bits at a time. Most modern computer systems use a 64-bit address bus, although the software which runs on the computer only uses a maximum of 32 bits at a time (known as 32-bit software).

Parallel busses are normally faster than serial busses (as they can transmit more bits in a single operation), but require many more lines (thus requiring more wires in the cable). A parallel data bus normally requires extra data handshaking lines to synchronise the flow of data between devices. Serial data transmission normally uses a start and end bit sequence to define the start and end of transmission. Figure 2.3 illustrates the differences between serial and parallel data busses. Parallel busses are typically used for local busses, or where there are no problems with cables with a relatively large number of wires. Typically, parallel busses are SCSI and IDE which are used to connect to hard disk drives, and typical serial busses are RS-232, and the USB.

Serial communications can operate at very high transmission rates; the main limiting factor is the transmission channel and the transmitter/receiver electronics. Gigabit Ethernet, for example, uses a transmission rate of 1 Gbps (125 MB/s) over high-quality twisted-pair copper cables, or over fibre optic cables (although this is a theoretical rate as more than one bit is sent at a time). For a 32-bit parallel bus, this would require a clocking rate of only 31.25 MHz (which requires much lower quality connectors and cables than the equivalent serial interface).

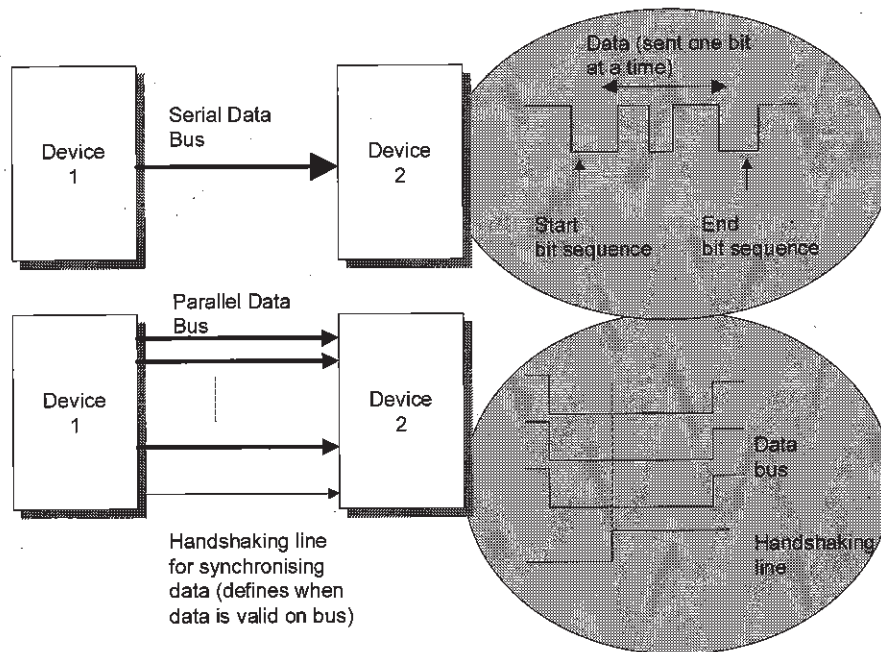


Figure 2.3 Serial/parallel data busses

Data transfer rates

The amount of data that a system can transfer at a time is normally defined either in bits per second (bps) or bytes per second (B/s). The more bytes (or bits) that can be transferred the faster the transfer will be. Typically serial busses are defined in bps, whereas parallel busses use B/s.

The transfer of the data occurs at regular intervals, which is defined by the period of the transfer clock. This period is either defined as a time interval (in seconds), or as a frequency (in Hz). For example, if a clock operates at a rate of 1 000 000 cycles per second, its frequency is 1 MHz, and its time interval will be one millionth of a second (1×10^{-6} s).

In general, if f is the clock frequency (in Hz), then the clock period (in seconds) will be

$$T = \frac{1}{f} \text{ sec}$$

Conversion from clock frequency to clock time interval

For example, if the clock frequency is 8 MHz, then the clock period will be:

$$\begin{aligned} T &= \frac{1}{8 \times 10^6} = 0.000000125 \text{ sec} \\ &= 0.125 \mu\text{s} \end{aligned}$$

Example of a calculation of clock time interval from clock frequency

The data transfer rate (in bits/second) is defined as:

$$\text{Data transfer rate (bps)} = \frac{\text{Number of bits transmitted per operation (bits)}}{\text{Transfer time per operation (s)}}$$

If operated with a fixed clock frequency for each operation then the data transfer rate (in bits/second) will be

$$\text{Data transfer rate (bps)} = \text{Number of bits transmitted per operation (bits)} \times \text{Clocking rate (Hz)}$$

For example, the ISA bus uses an 8MHz (8×10^6 Hz) clocking frequency and has a 16-bit data bus. Thus the maximum data transfer rate (in bps) will be:

$$\text{Data transfer rate} = 16 \times 8 \times 10^6 = 128 \times 10^6 \text{ b/s} = 128 \text{ Mbps}$$

Often it is required that the data rate is given in B/s, rather than bps. To convert from bps to B/s, eight divides the bps value. Thus to convert 128Mbps to B/s

$$\begin{aligned} \text{Data transfer rate} &= 128 \text{ Mbps} \\ &= \frac{128}{8} \text{ Mbps} = 16 \text{ MB/s} \end{aligned}$$

Example conversion from bps to B/s

For serial communication, if the time to transmit a single bit is $104.167 \mu\text{s}$ then the maximum data rate will be

$$\text{Data transfer rate} = \frac{1}{104.167 \times 10^{-6}} = 9600 \text{ bps}$$

Example conversion to bps for a serial transmission with a given transfer time interval

2.1.3 Address bus

The address bus is responsible for identifying the location into which the data is to be passed into. Each location in memory typically contains a single byte (8 bits), but could also be arranged as words (16 bits), or long words (32 bits). Byte-oriented memory is the most flexible as it also enables access to any multiple of eight bits. The size of the address bus thus indicates the maximum addressable number of bytes. Table 2.3 shows the size of addressable memory for a given address bus size. The number of addressable bytes is given by:

$$\text{Addressable locations} = 2^n \text{ bytes}$$

Addressable locations for a given address bus size

where n is the number of bits in the address bus. For example:

- A 1-bit address bus can address up to two locations (that is 0 and 1).
- A 2-bit address bus can address 2^2 or 4 locations (that is 00, 01, 10 and 11).
- A 20-bit address bus can address up to 2^{20} addresses (1 MB).
- A 32-bit address bus can address up to 2^{32} addresses (4 GB).

The units used for computers for defining memory are B (Bytes), kB (kiloBytes), MB (megaBytes) and GB (gigabytes). These are defined as:

- **KB** (kiloByte). This is defined as 2^{10} bytes, which is 1024 B.
- **MB** (megaByte). This is defined as 2^{20} bytes, which is 1024 kB, or 1 048 576 bytes.
- **GB** (gigaByte). This is defined as 2^{30} bytes, which is 1024 MB, or 1 048 576 kB, or 1 073 741 824 B.

Table 2.1 gives a table with addressable space for given address bus sizes.

Table 2.1 Addressable memory (in bytes) related to address bus size

Address bus size	Addressable memory (bytes)	Address bus size	Addressable memory (bytes)
1	2	15	32 K
2	4	16	64 K
3	8	17	128 K
4	16	18	256 K
5	32	19	512 K
6	64	20	1 M†
7	128	21	2 M
8	256	22	4 M
9	512	23	8 M
10	1 K*	24	16 M
11	2 K	25	32 M
12	4 K	26	64 M
13	8 K	32	4 G‡
14	16 K	64	16 GG

* 1 K represents 1024

† 1 M represents 1 048 576 (1024 K)

‡ 1 G represents 1 073 741 824 (1024 M)

Data handshaking

Handshaking lines are also required to allow the orderly flow of data. This is illustrated in Figure 2.4. Normally there are several different types of busses which connect to the system, these different busses are interfaced to with a bridge, which provides for the conversion between one type of bus and another. Sometimes devices connect directly onto the processor's bus; this is called a local bus, and is used to provide a fast interface with direct access without any conversions.

The most basic type of handshaking has two lines:

- Sending identification line – this identifies that a device is ready to send data.
- Receiving identification line – this identifies that device is a device is ready to receive data, or not.

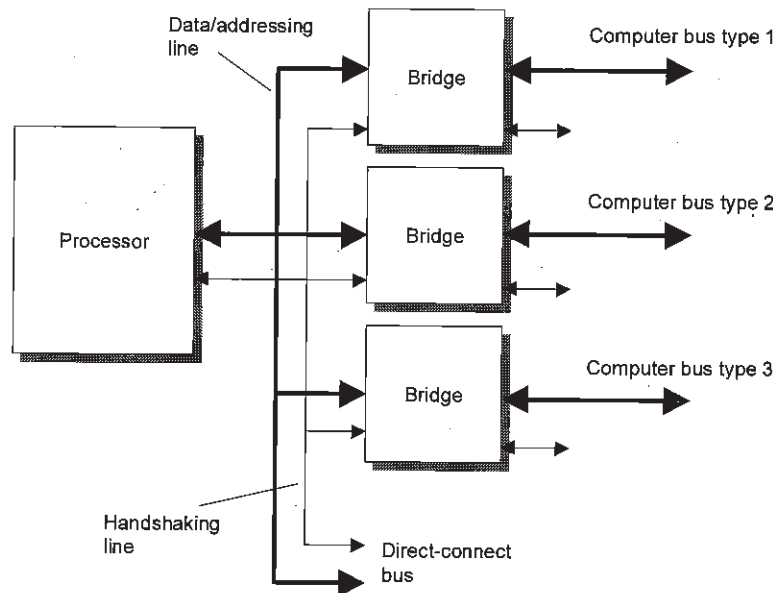


Figure 2.4 Computer bus connections

Figure 2.5 shows a simple form of handshaking of data, from Device 1 to Device 2. The sending status is identified by READY? and the receiving status by STATUS. Normally an event is identified by a signal line moving from one state to another, this is described as edge-triggered (rather than level-triggered where the actual level of the signal identifies its state). In the example in Figure 2.5, initially Device 1 puts data on the data bus, and identifies that it is ready to send data by changing the READY? line from a LOW to a HIGH level. Device 2 then identifies that it is reading the data by changing its STATUS line from a LOW to a HIGH. Next it identifies that it has read the data by changing the STATUS line from a HIGH to a LOW. Device 1 can then put new data on the data bus and start the cycle again by changing the READY? line from a LOW to a HIGH.

This type of communication only allows communication in one direction (from Device 1 to Device 2) and is known as simplex communications. The main types of communication are:

- **Simplex communication.** Only one device can communicate with the other, and thus only requires handshaking lines for one direction.
- **Half-duplex communication.** This allows communications from one device to the other, in any direction, and thus requires handshaking lines for either direction.
- **Full-duplex communications.** This allows communication from one device to another, in either direction, at the same time. A good example of this is in a telephone system, where a caller can send and receive at the same time. This requires separate transmit and receive data lines, and separate handshaking lines for either direction.

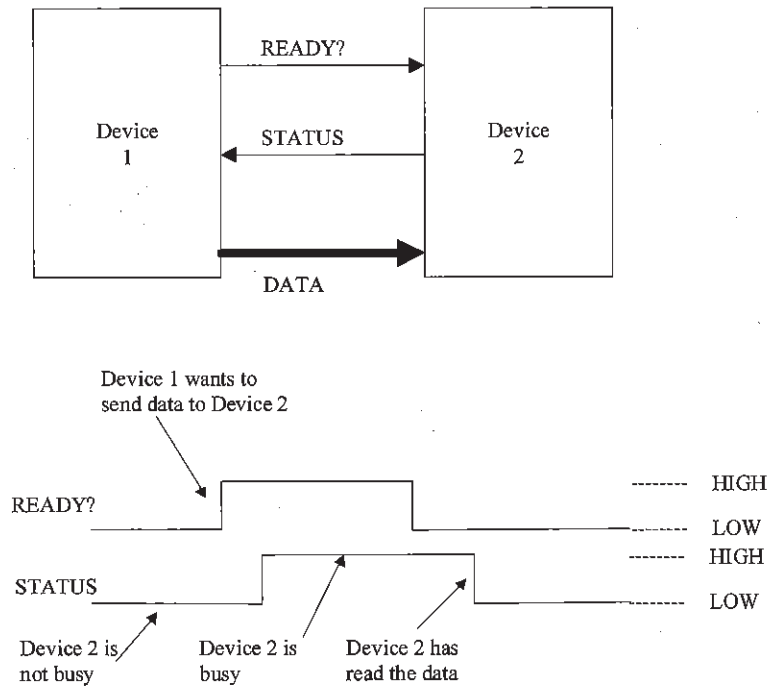


Figure 2.5 Simple handshaking of data

Control lines

Control lines define the operation of the data transaction, such as:

- Data flow direction – this identifies that data is either being read from a device or written to a device.
- Memory addressing type – this is typically either by identifying that the address access is direct memory accessing or indirect memory access. This identifies that the address on the bus is either a real memory location or is an address tag.
- Device arbitration – this identifies which device has control of the bus, and is typically used when there are many devices connected to a common bus, and any of the devices are allowed to communicate with any other of the devices on the bus.

2.1.4 Cables

The cable type used to transmit the data over the bus depends on several parameters, including:

- The signal bandwidth.
- The reliability of the cable.
- The maximum length between nodes.
- The possibility of electrical hazards.
- Power loss in the cables.

- Tolerance to harsh conditions.
- Expense and general availability of the cable.
- Ease of connection and maintenance.
- Ease of running cables, and so on.

The main types of cables used are standard copper cable, unshielded twisted-pair copper (UTP), shielded twisted-pair cable (STP), coaxial and fibre optic. Twisted-pair and coaxial cables transmit electric signals, whereas fibre optic cables transmit light pulses. Twisted-pair cables are not shielded and thus interfere with nearby cables. Public telephone lines generally use twisted-pair cables. In LANs they are generally used up to bit rates of 10 Mbps and with maximum lengths of 100 m.

Coaxial cable has a grounded metal sheath around the signal conductor. This limits the amount of interference between cables and thus allows higher data rates. Typically, they are used at bit rates of 100 Mbps for maximum lengths of 1 km.

The highest specification of the three cables is fibre optic. This type of cable allows extremely high bit rates over long distances. Fibre optic cables do not interfere with nearby cables and give greater security, give more protection from electrical damage by external equipment and greater resistance to harsh environments; they are also safer in hazardous environments.

Cable characteristics

The main characteristics of cables are attenuation, cross-talk and characteristic impedance. Attenuation defines the reduction in the signal strength at a given frequency for a defined distance. It is normally defined in dB/100 m, which is the attenuation (in dB) for 100 m. An attenuation of 3 dB/100 m gives a signal voltage reduction of 0.5 for every 100 m. Table 2.2 lists some attenuation rates and equivalent voltage ratios; they are illustrated in Figure 2.6. Attenuation is given by

$$\text{Attenuation} = 20 \log_{10} \left(\frac{V_{\text{in}}}{V_{\text{out}}} \right) \text{ dB}$$

Calculation of attenuation from input and output voltages

For example if the input voltage to a cable is 10 V and the voltage at the other end is only 7 V, then the attenuation is calculated as

$$\text{Attenuation} = 20 \log_{10} \left(\frac{10}{7} \right) = 3.1 \text{ dB}$$

Coaxial cables have an inner core separated from an outer shield by a dielectric. They have an accurate characteristic impedance (which reduces reflections), and because they are shielded they have very low cross-talk levels. They tend also to have very low attenuation, (such as 1.2 dB at 4 MHz), with a relatively flat response. UTPs (unshielded twisted-pair cables) have either solid cores (for long cable runs) or are stranded patch cables (for short runs, such as connecting to workstations, patch panels, and so on). Solid cables should not be flexed, bent or twisted repeatedly, whereas stranded cable can be flexed without damaging the cable. Coaxial cables use BNC connectors while UTP cables use either the RJ-11 (small

connector which is used to connect the handset to the telephone) or the RJ-45 (larger connector which is typically used in networked applications to connect a network adapter to a network hub).

The characteristic impedance of a cable and its connectors are important, as all parts of the transmission system need to be matched to the same impedance. This impedance is normally classified as the characteristic impedance of the cable. Any differences in the matching result in a reduction of signal power and produce signal reflections (or ghosting).

Cross-talk is important as it defines the amount of signal that crosses from one signal path to another. This causes some of the transmitted signal to be received back where it was transmitted. Capacitance (pF/100 m) defines the amount of distortion in the signal caused by each signal pair. The lower the capacitance value, the lower the distortion.

Table 2.2 Attenuation rates as a ratio

<i>dB</i>	<i>Ratio</i>	<i>dB</i>	<i>Ratio</i>	<i>dB</i>	<i>Ratio</i>
0	1.000	10	0.316	60	0.001
1	0.891	15	0.178	65	0.0006
2	0.794	20	0.100	70	0.0003
3	0.708	25	0.056	75	0.0002
4	0.631	30	0.032	80	0.0001
5	0.562	35	0.018	85	0.00006
6	0.501	40	0.010	90	0.00003
7	0.447	45	0.0056	95	0.00002
8	0.398	50	0.0032	100	0.00001
9	0.355	55	0.0018		

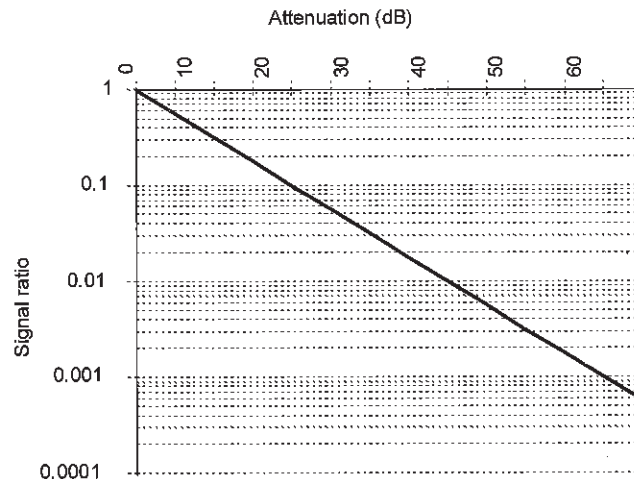


Figure 2.6 Signal ratio related to attenuation

Typical cables used are:

- Coaxial cable – cables with an inner core and a conducting shield having characteristic impedance of either $75\ \Omega$ for TV signal or $50\ \Omega$ for other types.
- Cat-3 UTP cable – level 3 cables have non-twisted-pair cores with a characteristic impedance of $100\ \Omega$ ($\pm 15\ \Omega$) and a capacitance of $59\ \text{pF/m}$. Conductor resistance is around $9.2\ \Omega/100\ \text{m}$.
- Cat-5 UTP cable – level 5 cables have twisted-pair cores with a characteristic impedance of $100\ \Omega$ ($\pm 15\ \Omega$) and a capacitance of $45.9\ \text{pF/m}$. Conductor resistance is around $9\ \Omega/100\ \text{m}$.

The Electrical Industries Association (EIA) has defined five main types of cables. Levels 1 and 2 are used for voice and low-speed communications (up to 4 Mbps). Level 3 is designed for LAN data transmission up to 16 Mbps and level 4 is designed for speeds up to 20 Mbps. Level 5 cables, have the highest specification of the UTP cables and allow data speeds of up to 100 Mbps. The main EIA specification on these types of cables is EIA/TIA568 and the ISO standard is ISO/IEC11801.

Table 2.3 gives typical attenuation rates (dB/100 m) for Cat-3, Cat-4 and Cat-5 cables. Notice that the attenuation rates for Cat-4 and Cat-5 are approximately the same. These two types of cable have lower attenuation rates than equivalent Cat-3 cables. Notice that the attenuation of the cable increases as the frequency increases. This is due to several factors, such as the skin effect, where the electrical current in the conductors becomes concentrated around the outside of the conductor, and the fact that the insulation (or dielectric) between the conductors actually starts to conduct as the frequency increases.

The Cat-3 cable produces considerable attenuation over a distance of 100 m. The table shows that the signal ratio of the output to the input at 1 MHz, will be 0.76 (2.39 dB), then, at 4 MHz it is 0.55 (5.24 dB), until at 16 MHz it is 0.26. This differing attenuation at different frequencies produces not just a reduction in the signal strength but also distorts the signal (because each frequency is affected differently by the cable. Cat-4 and Cat-5 cables also produce distortion but their effects will be lessened because attenuation characteristics have flatter shapes.

Table 2.4 gives typical near-end cross-talk rates (dB/100 m) for Cat-3, Cat-4 and Cat-5 cables. The higher the figure, the smaller the cross-talk. Notice that Cat-3 cables have the most cross-talk and Cat-5 have the least, for any given frequency. Notice also that the cross talk increases as the frequency of the signal increases. Thus, high-frequency signals have more cross-talk than lower-frequency signals.

Table 2.3 Attenuation rates (dB/100 m) for Cat-3, Cat-4 and Cat-5 cable

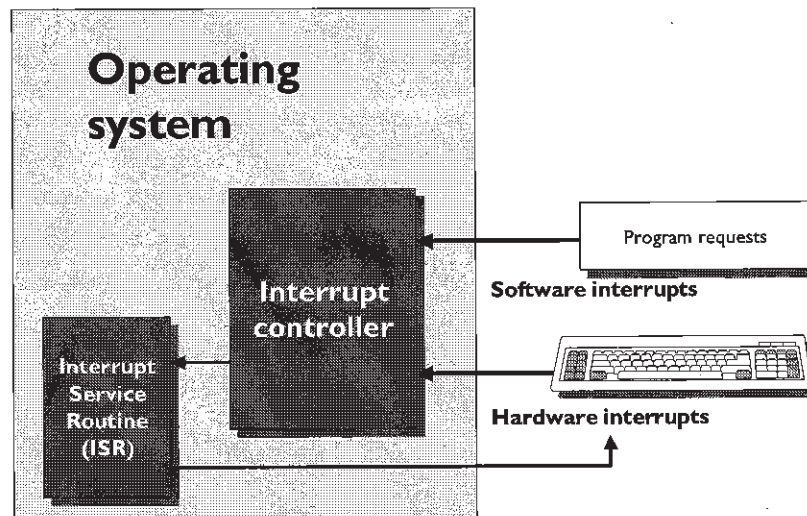
Frequency (MHz)	Attenuation rate (dB/100 m)		
	Cat-3	Cat-4	Cat-5
1	2.39	1.96	2.63
4	5.24	3.93	4.26
10	8.85	6.56	6.56
16	11.8	8.2	8.2

Table 2.4 Near-end cross-talk (dB/100 m) for Cat-3, Cat-4 and Cat-5 cable

Frequency (MHz)	Near end cross-talk (dB/100 m)		
	Cat-3	Cat-4	Cat-5
1	13.45	18.36	21.65
4	10.49	15.41	18.04
10	8.52	13.45	15.41
16	7.54	12.46	14.17

2.2 Interrupts

An interrupt allows a program or an external device to interrupt the execution of a program. The generation of an interrupt can occur by hardware (hardware interrupt) or software (software interrupt). When an interrupt occurs an interrupt service routine (ISR) is called. For a hardware interrupt the ISR then communicates with the device and processes any data. When it has finished the program execution returns to the original program. A software interrupt causes the program to interrupt its execution and goes to an interrupt service routine. Typical software interrupts include reading a key from the keyboard, outputting text to the screen and reading the current date and time. The operating system must respond to interrupts from external devices, as illustrated in Figure 2.7.

**Figure 2.7** Interrupt service routine

2.2.1 Software interrupts

BIOS and the operating system

The Basic Input/Output System (BIOS) communicates directly with the hardware of the computer. It consists of a set of programs which interface with devices such as keyboards, displays, printers, serial ports and disk drives. These programs allow the user to write application programs that contain calls to these functions, without having to worry about controlling them or which type of equipment is being used. Without BIOS, the computer system would simply consist of a bundle of wires and electronic devices.

There are two main parts to BIOS. The first is the part permanently stored in a ROM (the ROM BIOS). It is this part that starts the computer (or bootstrap) and contains programs which communicate with resident devices. The second stage is loaded when the operating system is started. This part is non-permanent.

An operating system allows the user to access the hardware in an easy-to-use manner. It accepts commands from the keyboard and displays them to the monitor. The Disk Operating System, or DOS, gained its name from its original purpose of providing a controller for the computer to access its disk drives. The language of DOS consists of a set of commands which are entered directly by the user and are interpreted to perform file management tasks, program execution and system configuration. It makes calls to BIOS to execute these. The main functions of DOS are to run programs, copy and remove files, create directories, move within a directory structure and to list files. Microsoft Windows calls BIOS programs directly.

Interrupt vectors

Interrupt vectors are addresses which inform the interrupt handler as to where to find the ISR. All interrupts are assigned a number from 0 to 255. The interrupt vectors associated with each interrupt number are stored in the lower 1024 bytes of PC memory. For example, interrupt 0 is stored from 0000:0000 to 0000:0003, interrupt 1 from 0000:0004 to 0000:0007, and so on. The first two bytes store the offset and the next two store the segment address. Each interrupt number is assigned a predetermined task, as outlined in Table 2.5. An interrupt can be generated either by external hardware, software, or by the processor. Interrupts 0, 1, 3, 4, 6 and 7 are generated by the processor. Interrupts from 8 to 15 and interrupt 2 are generated by external hardware. These get the attention of the processor by activating a interrupt request (IRQ) line. The IRQ0 line connects to the system timer, the keyboard to IRQ1, and so on. Most other interrupts are generated by software.

Processor interrupts

The processor-generated interrupts normally occur either when a program causes a certain type of error or if it is being used in a debug mode. In the debug mode the program can be made to break from its execution when a break-point occurs. This allows the user to test the status of the computer. It can also be forced to step through a program one operation at a time (single-step mode).

Table 2.5 Interrupt handling (codes followed by 'h' are in hexadecimal)

<i>Interrupt</i>	<i>Name</i>	<i>Generated by</i>
00 (00h)	Divide error	processor
01 (00h)	Single step	processor
02 (02h)	Non-maskable interrupt	external equipment
03 (03h)	Breakpoint	processor
04 (04h)	Overflow	processor
05 (05h)	Print screen	Shift-Print screen key stroke
06 (06h)	Reserved	processor
07 (07h)	Reserved	processor
08 (08h)	System timer	hardware via IRQ0
09 (09h)	Keyboard	hardware via IRQ1
10 (0Ah)	Reserved	hardware via IRQ2
11 (0Bh)	Serial communications (COM2)	hardware via IRQ3
12 (0Ch)	Serial communications (COM1)	hardware via IRQ4
13 (0Dh)	Reserved	hardware via IRQ5
14 (0Eh)	Floppy disk controller	hardware via IRQ6
15 (0Fh)	Parallel printer	hardware via IRQ7
16 (10h)	BIOS – Video access	software
17 (11h)	BIOS – Equipment check	software
18 (12h)	BIOS – Memory size	software
19 (13h)	BIOS – Disk operations	software
20 (14h)	BIOS – Serial communications	software
22 (16h)	BIOS – Keyboard	software
23 (17h)	BIOS – Printer	software
25 (19h)	BIOS – Reboot	software
26 (1Ah)	BIOS – Time of day	software
28 (1Ch)	BIOS – Ticker timer	software
33 (21h)	DOS – DOS services	software
39 (27h)	DOS – Terminate and stay resident	software

2.2.2 Hardware interrupts

Computer systems either use polling or interrupt-driven software to service external equipment. With polling the computer continually monitors a status line and waits for it to become active, whereas an interrupt-driven device sends an interrupt request to the computer, which is then serviced by an interrupt service routine (ISR). Interrupt-driven devices are normally better in that the computer is thus free to do other things, whereas polling slows the system down as it must continually monitor the external device. Polling can also cause problems in that a device may be ready to send data and the computer is not watching the status line at that point. Figure 2.8 illustrates polling and interrupt-driven devices.

The generation of an interrupt can occur by hardware or software, as illustrated in Figure 2.9. If a device wishes to interrupt the processor, it informs the programmable interrupt controller (PIC). The PIC then decides whether it should interrupt the processor. If there is a processor interrupt then the processor reads the PIC to determine which device caused the interrupt. Then, depending on the device that caused the interrupt, a call to an ISR is made. The ISR then communicates with the device and processes any data. When it has finished the program execution returns to the original program.

A software interrupt causes the program to interrupt its execution and goes to an interrupt service routine. Typical software interrupts include reading a key from the keyboard, output-

ting text to the screen and reading the current date and time.

Hardware interrupts allow external devices to gain the attention of the processor. Depending on the type of interrupt the processor leaves the current program and goes to a special program called an interrupt service routine (ISR). This program communicates with the device and processes any data. After it has completed its task then program execution returns to the program that was running before the interrupt occurred. Examples of interrupts include the processing of keys from a keyboard and data from a sound card.

As previously mentioned, a device informs the processor that it wants to interrupt it by setting an interrupt line on the PC. Then, depending on the device that caused the interrupt, a call to an ISR is made. Each PIC allows access to eight interrupt request lines. Most PCs use two PICs which gives access to 16 interrupt lines.

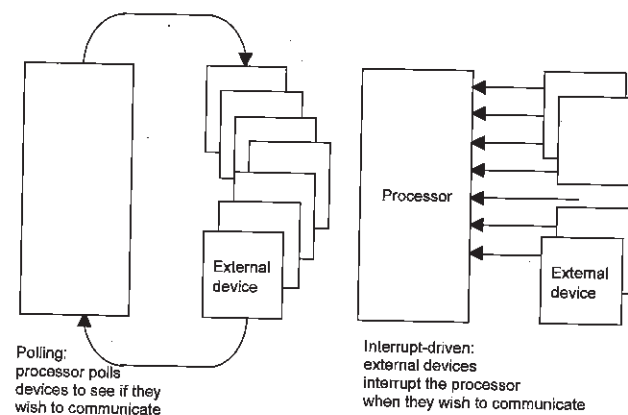


Figure 2.8 Polling or interrupt-driven communications

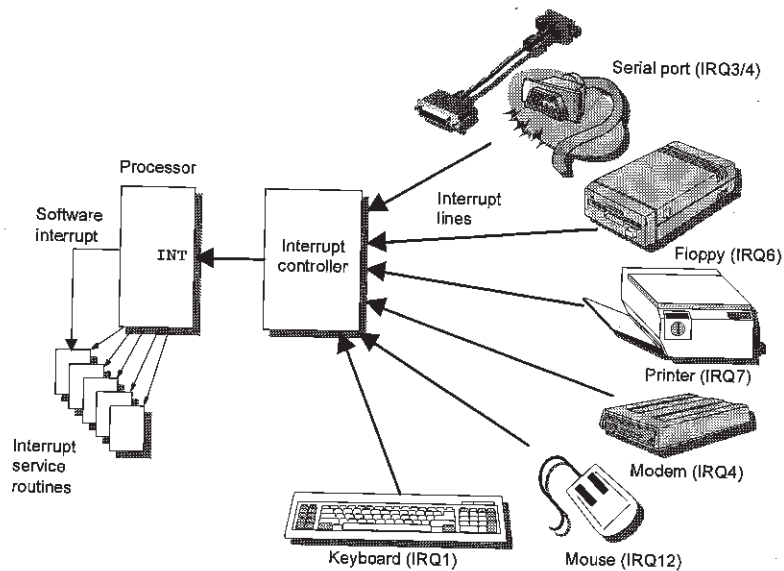


Figure 2.9 Interrupt handling

Interrupt vectors

Each device that requires to be 'interrupt-driven' is assigned an IRQ (interrupt request) line. Each IRQ is active high. The first eight (IRQ0–IRQ7) map into interrupts 8 to 15 (08h–0Fh) and the next eight (IRQ8–IRQ15) into interrupts 112 to 119 (70h–77h). Table 2.6 outlines the usage of each of these interrupts. When IRQ0 is made active, the ISR corresponds to interrupt vector 8. IRQ0 normally connects to the system timer, the keyboard to IRQ1, and so on. The standard set-up of these interrupts is illustrated in Figure 2.10. The system timer interrupts the processor 18.2 times per second and is used to update the system time. When the keyboard has data, it interrupts the processor with the IRQ1 line.

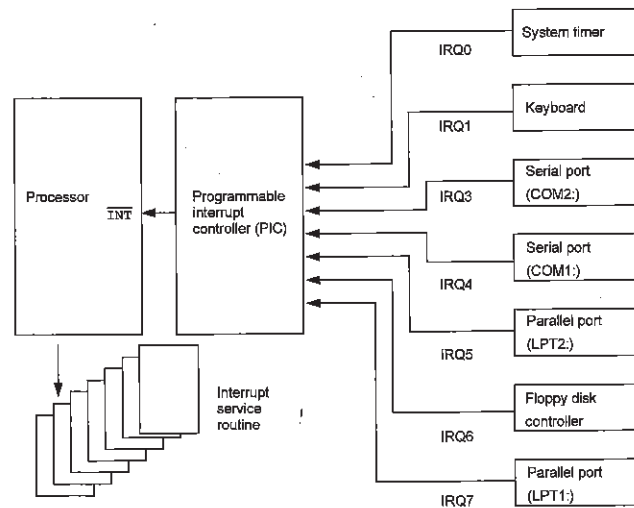


Figure 2.10 Standard usage of IRQ lines

Table 2.6 Interrupt handling

Interrupt	Name	Generated by
08 (08h)	System timer	IRQ0
09 (09h)	Keyboard	IRQ1
10 (0Ah)	Reserved	IRQ2
11 (0Bh)	Serial communications (COM2:)	IRQ3
12 (0Ch)	Serial communications (COM1:)	IRQ4
13 (0Dh)	Parallel port (LPT2:)	IRQ5
14 (0Eh)	Floppy disk controller	IRQ6
15 (0Fh)	Parallel printer (LPT1:)	IRQ7
112 (70h)	Real-time clock	IRQ8
113 (71h)	Redirection of IRQ2	IRQ9
114 (72h)	Reserved	IRQ10
115 (73h)	Reserved	IRQ11
116 (74h)	Reserved	IRQ12
117 (75h)	Math co-processor	IRQ13
118 (76h)	Hard disk controller	IRQ14
119 (77h)	Reserved	IRQ15

Data received from serial ports interrupts the processor with IRQ3 and IRQ4 and the parallel ports use IRQ5 and IRQ7. If one of the parallel, or serial, ports does not exist then the IRQ line normally assigned to it can be used by another device. It is typical for interrupt-driven I/O cards, such as a sound card, to have a programmable IRQ line which is mapped to an IRQ line that is not being used.

Note that several devices can use the same interrupt line. A typical example is COM1 : and COM3 : sharing IRQ4 and COM2 : and COM4 : sharing IRQ3. If they do share then the ISR must be able to poll the shared devices to determine which of them caused the interrupt. If two different types of device (such as a sound card and a serial port) use the same IRQ line then there may be a contention problem as the ISR may not be able to communicate with different types of interfaces.

Figure 2.11 shows a sample window displaying interrupt usage. In this case it can be seen that the system timer uses IRQ0, the keyboard uses IRQ1, the PIC uses IRQ2, and so on. Notice that a sound blaster is using IRQ5. This interrupt is normally reserved for the secondary printer port. If there is no printer connected then IRQ5 can be used by another device. Some devices can have their I/O address and interrupt line changed. An example is given in Figure 2.12. In this case, the IRQ line is set to IRQ7 and the base address is 378h.

Typical uses of interrupts are:

IRQ0: System timer

The system timer uses IRQ0 to interrupt the processor 18.2 times per second and is used to keep the time-of-day clock updated.

IRQ1: Keyboard data ready

The keyboard uses IRQ1 to signal to the processor that data is ready to be received from the keyboard. This data is normally a scan code.

IRQ2: Redirection of IRQ9

The BIOS redirects the interrupt for IRQ9 back here.

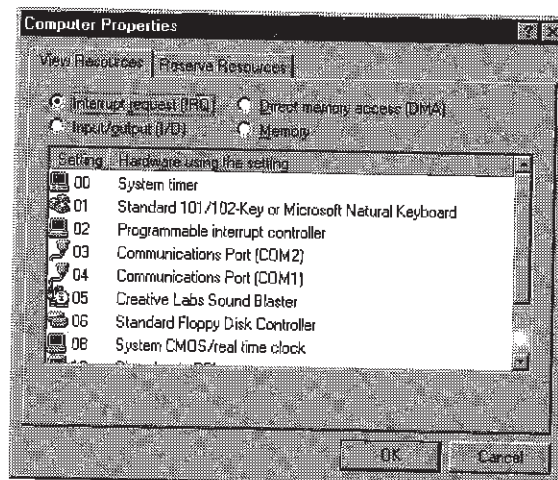


Figure 2.11 Standard usage of IRQ lines

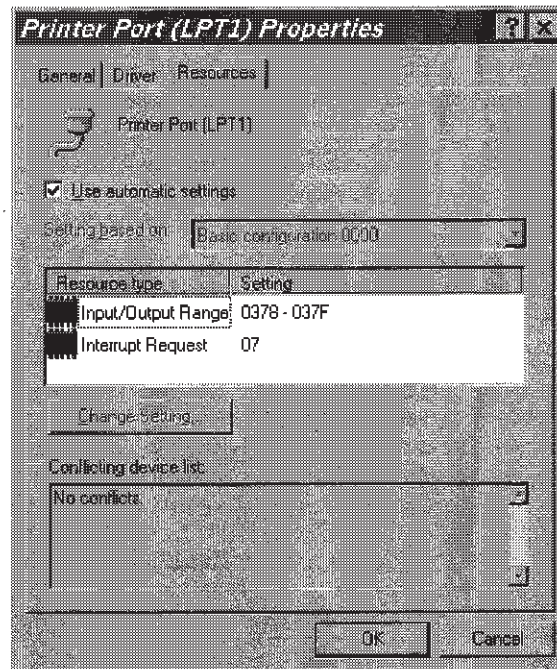


Figure 2.12 Standard set-up of IRQ lines

- | | |
|--|--|
| IRQ3: Secondary serial port (COM2:) | The secondary serial port (COM2:) uses IRQ3 to interrupt the processor. Typically, COM3: to COM8: also use it, although COM3: may use IRQ4. |
| IRQ4: Primary serial port (COM1:) | The primary serial port (COM1:) uses IRQ4 to interrupt the processor. Typically, COM3: also uses it. |
| IRQ5: Secondary parallel port (LPT2:) | On older PCs the IRQ5 line was used by the fixed disk. On newer systems the secondary parallel port uses it. Typically, it is used by a sound card on PCs which have no secondary parallel port connected. |
| IRQ6: Floppy disk controller | The floppy disk controller activates the IRQ6 line on completion of a disk operation. |
| IRQ7: Primary parallel port (LPT1:) | Printers (or other parallel devices) activate the IRQ7 line when they become active. As with IRQ5 it may be used by another device, if there are no other devices connected to this line. |
| IRQ9 | Redirected to IRQ2 service routine. |

Programmable interrupt controller (PIC)

The PC uses the 8259 PIC to control hardware-generated interrupts. It is known as a programmable interrupt controller and has eight input interrupt request lines and an output line to secondary PIC are then assigned IRQ lines of IRQ8 to IRQ15. This set-up is shown in Figure 2.13. When an interrupt occurs on any of these lines it is sensed by the processor on interrupt the processor. Originally, PCs only had one PIC and eight IRQ lines (IRQ0-IRQ7). Modern PCs can use up to 15 IRQ lines which are set up by connecting a secondary PIC interrupt request output line to the IRQ2 line of the primary PIC. The interrupt lines on the IRQ2 line. The processor then interrogates the primary and secondary PIC for the interrupt line which caused the interrupt.

The primary and secondary PICs are programmed via port addresses 20h and 21h, as given in Table 2.7. The operation of the PIC is programmed using registers. The IRQ input lines are either configured as level-sensitive or edge-triggered interrupt. With edge-triggered interrupts, a change from a low to a high on the IRQ line causes the interrupt. A level-sensitive interrupt occurs when the IRQ line is high. Most devices generate edge-triggered interrupts.

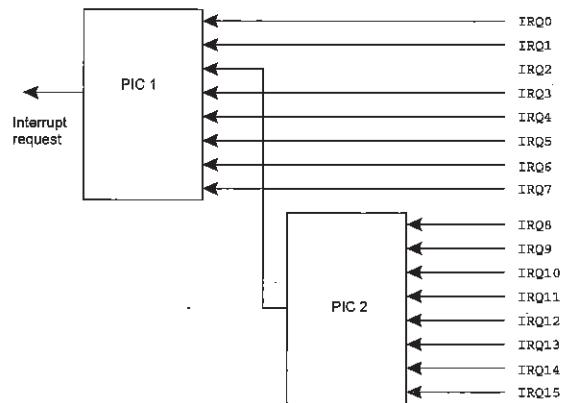


Figure 2.13 PC PIC connections

In the IMR an interrupt line is enabled by setting the assigned bit to a 0 (zero). This allows the interrupt line to interrupt the processor. Figure 2.14 shows the bit definitions of the IMR. For example, if bit 0 is set to a zero then the system timer on IRQ0 is enabled.

Table 2.7 Interrupt port addresses

Port address	Name	Description
20h	Interrupt control register (ICR)	Controls interrupts and signifies the end of an interrupt
21h	Interrupt mask register (IMR)	Used to enable and disable interrupt lines

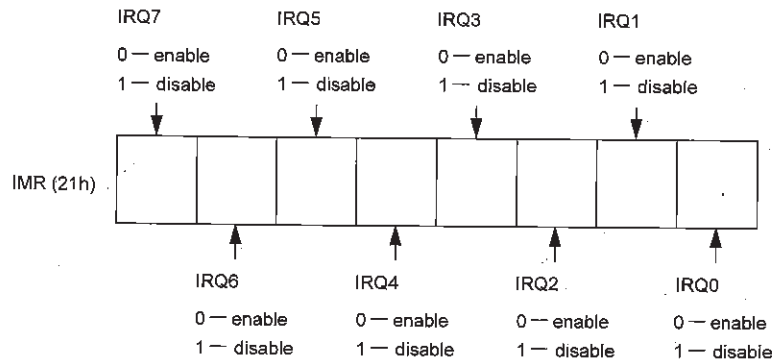


Figure 2.14 Interrupt mask register bit definitions

In the example code given next the lines IRQ0, IRQ1 and IRQ6 are allowed to interrupt the processor, whereas, IRQ2, IRQ3, IRQ4 and IRQ7 are disabled:

```
_outp(0x21)=0xBC; /* 1011 1100 enable disk
                    (bit 6), keyboard (1) and timer (0) interrupts */
```

When an interrupt occurs all other interrupts are disabled and no other device can interrupt the processor. Interrupts are enabled again by setting the EOI bit on the interrupt control port, as shown in Figure 2.15.

The following code enables interrupts:

```
_outp(0x20,0x20); /* EOI command */
```

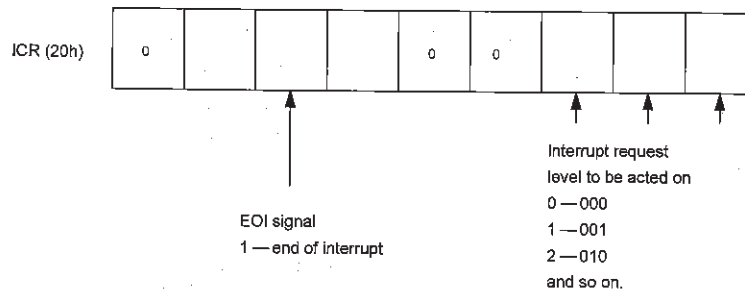


Figure 2.15 Interrupt control register bit definitions

2.3 Interfacing

There are two main methods of communicating with external equipment, either they are mapped into the physical memory and given a real address on the address bus (memory mapped I/O) or they are mapped into a special area of input/output memory (isolated I/O).

Figure 2.16 shows the two methods. Devices mapped into memory are accessed by reading or writing to the physical address. Isolated I/O provides ports which are gateways between the interface device and the processor. They are isolated from the system using a buffering system and are accessed by four machine code instructions. The IN instruction inputs a byte, or a word, and the OUT instruction outputs a byte, or a word. A high-level compiler interprets the equivalent high-level functions and produces machine code which uses these instructions.

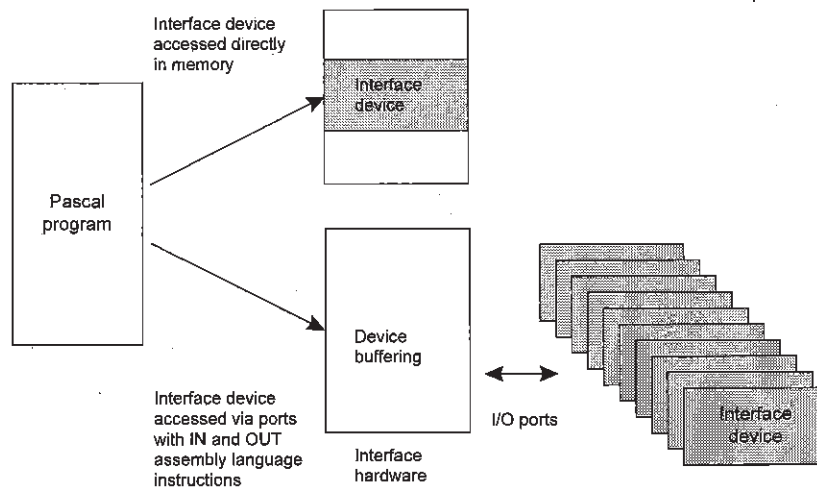


Figure 2.16 Memory mapping or isolated interfacing

2.3.1 Interfacing with memory

The 80x86 processor interfaces with memory through a bus controller, as shown in Figure 2.17. This device interprets the microprocessor signals and generates the required memory signals. Two main output lines differentiate between a read or a write operation ($R/\overline{W}$) and between direct and isolated memory access ($M/\overline{IO}$). The $R/\overline{W}$ line is low when data is being written to memory and high when data is being read. When $M/\overline{IO}$ is high, direct memory access is selected and when low, the isolated memory is selected.

2.3.2 Memory mapped I/O

Interface devices can map directly onto the system address and data bus. In a PC-compatible system the address bus is 20 bits wide, from address 00000h to FFFFFh (1 MB). If the PC is being used in an enhanced mode (such as with Microsoft Windows) it can access the area of memory above 1 MB. If it uses 16-bit software (such as Microsoft Windows 3.1) then it can address up to 16 MB of physical memory, from 000000h to FFFFFFFh. If it uses 32-bit software (such as Microsoft Windows 95/98/NT/2000) then the software can address up to 4 GB of physical memory, from 00000000h to FFFFFFFFh. Figure 2.18 gives a typical memory allocation.

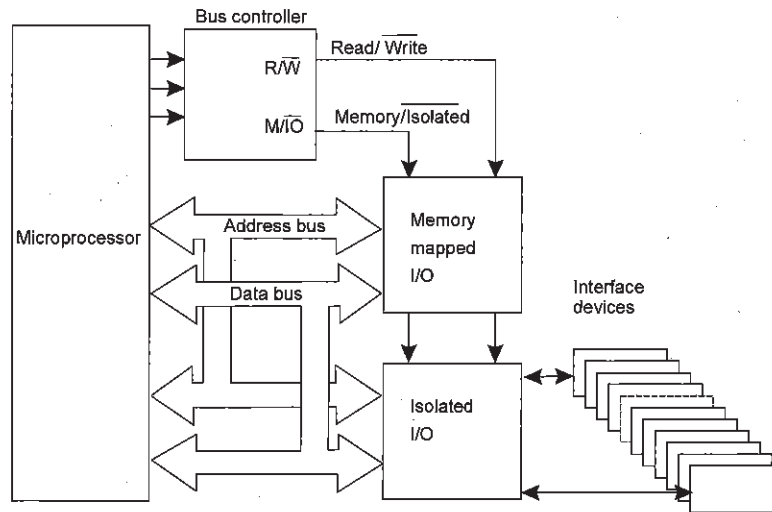


Figure 2.17 Access memory mapped and isolated I/O

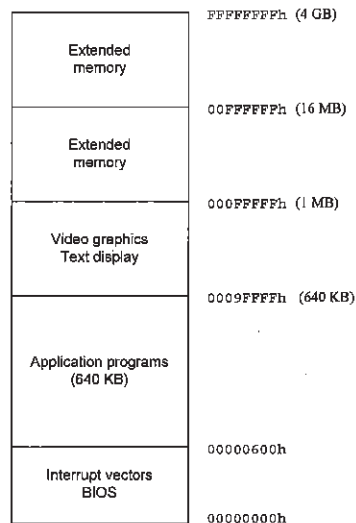


Figure 2.18 Typical PC memory map

2.3.3 Isolated I/O

Devices are not normally connected directly onto the address and data bus of the computer because they may use part of the memory that a program uses or they could cause a hardware fault. On modern PCs only the graphics adaptor is mapped directly into memory, the rest communicate through a specially reserved area of memory, known as isolated I/O memory.

Isolated I/O uses 16-bit addressing from 0000h to FFFFh, thus up to 64 KB of memory can be mapped. Figure 2.19 shows an example for a computer in the range from 0000h to 0064h and Figure 2.20 shows from 0378h to 03FFh. It can be seen from Figure 2.19 that the keyboard maps into addresses 0060h and 0064h, the speaker maps to address 0061h

and the system timer between 0040h and 0043h. Table 2.8 shows the typical uses of the isolated memory area.

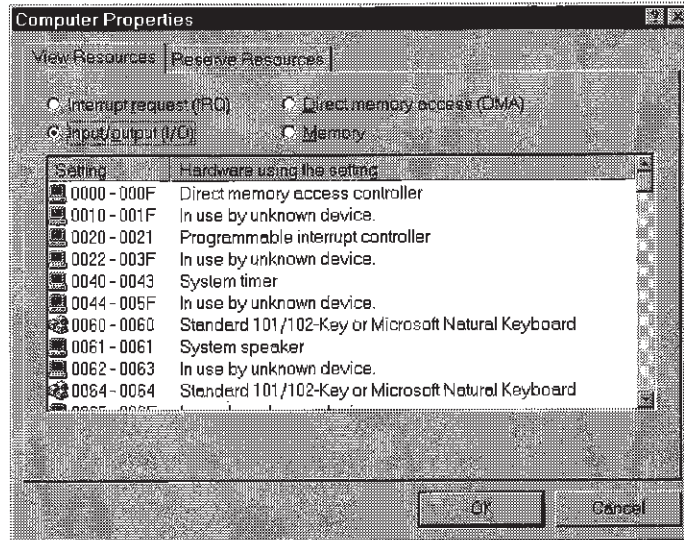


Figure 2.19 Example I/O memory map from 0000h to 0064h

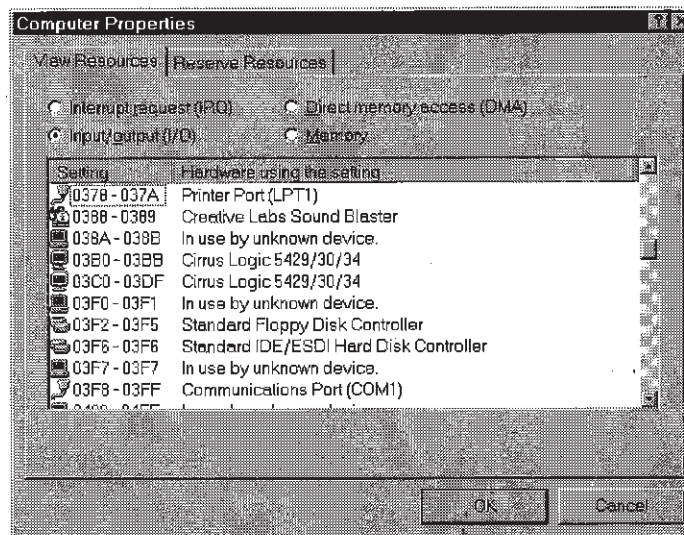


Figure 2.20 Example I/O memory map from 0378h to 03FFh

Table 2.8 Typical isolated I/O memory map

Address	Device
000h-01Fh	DMA controller
020h-021h	Programmable interrupt controller
040h-05Fh	Counter/Timer
060h-07Fh	Digital I/O
080h-09Fh	DMA controller
0A0h-0BFh	NMI reset
0C0h-0DFh	DMA controller
0E0h-0FFh	Math coprocessor
170h-178h	Hard disk (Secondary IDE drive or CD-ROM drive)
1F0h-1F8h	Hard disk (Primary IDE drive)
200h-20Fh	Game I/O adapter
210h-217h	Expansion unit
278h-27Fh	Second parallel port (LPT2:)
2F8h-2FFh	Second serial port (COM2:)
300h-31Fh	Prototype card
378h-37Fh	Primary parallel port (LPT1:)
380h-38Ch	SDLC interface
3A0h-3AFh	Primary binary synchronous port
3B0h-3BFh	Graphics adapter
3C0h-3DFh	Graphics adapter
3F0h-3F7h	Floppy disk controller
3F8h-3FFh	Primary serial port (COM1:)

Inputting a byte from an I/O port

The assembly language command to input a byte is

```
IN AL, DX
```

where DX is the data register which contains the address of the input port. The 8-bit value loaded from this address is put into the register A

For Turbo/Borland C the equivalent function is `inportb()`. Its general syntax is as follows:

```
value=inportb(PORTADDRESS);
```

where PORTADDRESS is the address of the input port and value is loaded with the 8-bit value from this address. This function is prototyped in the header file `dos.h`.

For Turbo Pascal the equivalent is accessed via the `port[]` array. Its general syntax is as follows:

```
value:=port[PORTADDRESS];
```

where PORTADDRESS is the address of the input port and value the 8-bit value at this address. To gain access to this function the statement uses `dos` requires to be placed near the top of the program.

Microsoft C++ uses the equivalent `_inp()` function (which is prototyped in `conio.h`).

Inputting a word from a port

The assembly language command to input a word is

```
IN AX, DX
```

where DX is the data register which contains the address of the input port. The 16-bit value loaded from this address is put into the register AX.

For Turbo/Borland C the equivalent function is `inport()`. Its general syntax is as follows:

```
value=inport(PORTADDRESS);
```

where PORTADDRESS is the address of the input port and value is loaded with the 16-bit value at this address. This function is prototyped in the header file `dos.h`.

For Turbo Pascal the equivalent is accessed via the `portw[]` array. Its general syntax is as follows:

```
value:=portw[PORTADDRESS];
```

where PORTADDRESS is the address of the input port and value is the 16-bit value at this address. To gain access to this function the statement uses `dos` requires to be placed near the top of the program.

Microsoft C++ uses the equivalent `_inpw()` function (which is prototyped in `conio.h`).

Outputting a byte to an I/O port

The assembly language command to output a byte is

```
OUT DX, AL
```

where DX is the data register which contains the address of the output port. The 8-bit value sent to this address is stored in register AL.

For Turbo/Borland C the equivalent function is `outportb()`. Its general syntax is as follows:

```
outportb(PORTADDRESS,value);
```

where PORTADDRESS is the address of the output port and value is the 8-bit value to be sent to this address. This function is prototyped in the header file `dos.h`.

For Turbo Pascal the equivalent is accessed via the `port[]` array. Its general syntax is as follows:

```
port[PORTADDRESS]:=value;
```

where PORTADDRESS is the address of the output port and value is the 8-bit value to be sent to that address. To gain access to this function the statement uses `dos` requires to be placed near the top of the program.

Microsoft C++ uses the equivalent `_outp()` function (which is prototyped in `conio.h`).

Outputting a word

The assembly language command to input a byte is:

```
OUT DX,AX
```

where DX is the data register which contains the address of the output port. The 16-bit value sent to this address is stored in register AX.

For Turbo/Borland C the equivalent function is `outport()`. Its general syntax is as follows:

```
outport(PORTADDRESS,value);
```

where PORTADDRESS is the address of the output port and value is the 16-bit value to be sent to that address. This function is prototyped in the header file `dos.h`.

For Turbo Pascal the equivalent is accessed via the `port[]` array. Its general syntax is as follows:

```
portw[PORTADDRESS]:=value;
```

where PORTADDRESS is the address of the output port and value is the 16-bit value to be sent to that address. To gain access to this function the statement uses `dos` requires to be placed near the top of the program.

Microsoft C++ uses the equivalent `_outp()` function (which is prototyped in `conio.h`).

In-line assembly language

Most modern C++ development systems use an inline assembler which allows assembly language code to be embedded with C++ code. This code can use any C variable or function name that is in scope. The `__asm` keyword invokes the inline assembler and can appear wherever a C statement is legal. The following code is a simple `__asm` block enclosed in braces.

```
__asm
{
    /* Initialize serial port */
    mov dx,0x01; /* COM2: */
    mov al,0xD2; /* serial port parameters */
    mov ah,0x0; /* initialize serial port */
    int 14h;
    line_status=ah;
    modem_status=al;
}
```

Note these statements can also be inserted after the `__asm` keyword, such as:

```
__asm mov dx,0x01; /* COM2: */
__asm mov al,0xD2; /* serial port parameters */
__asm mov ah,0x0; /* initialize serial port */
__asm int 14h;
__asm line_status=ah;
__asm modem_status=al;
```

2.4 PC Systems

In selecting a PC many different components must be considered, especially in the way that they connect. Figure 2.21 outlines some of the component parts and the decisions that have to be made on each component.

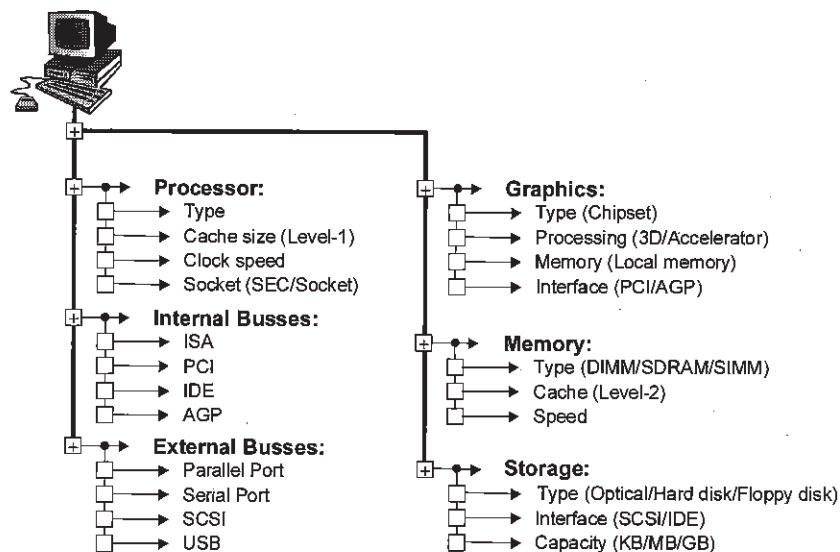


Figure 2.21 PC components

The Top 5 things that affect the *general* performance of a PC (in ranked order) are:

1. **Processor.** The type of the processor, its speed, its socket (which helps in upgrading in the future), its interface to Level-2 cache, and so on. Additionally, MMX™, (which is an Intel trademark, but many read it as MultiMedia eXtension) can speed-up multi-media applications.
2. **Local Memory.** Most operating systems can run multiple programs, each of which require their own memory space. When the system runs out of electronic memory (local memory), it uses the hard disk for an extra storage (to create a virtual memory). Hard disk accesses are much slower than electronic memory, thus the system is severely slowed down if there is a lack of local electronic memory. Most modern operating sys-

tems require a great deal of local electronic memory to operate.

3. **Graphics adaptor.** The graphics adaptor can be a major limiting factor on the performance of a system. New interfaces, such as AGP, considerably speed-up graphics performance. Another limiting factor is the amount of local memory on the graphics adaptor. The more memory, the higher the resolution that can be used, and the more colours that can be displayed. AGP is overcoming this limiting factor, as it allows the main electronic memory to be used to store graphics images.
4. **Cache capacity.** Cache memory has caused a great increase in the performance of a system. If a cache controller makes a correct guess, the processor merely has to examine the contents of the cache to get the required information. A level-1 cache is the fastest and is typically connected directly to the processor (normally inside the processor package), and the level-2 cache is on the motherboard.
5. **Hard disk capacity/interface.** The hard disk typically has an affect on the running of a program, as the program and its component parts must be loaded from the disk. The interface is thus extremely important as it defines the maximum data rate. SCSI has fast modes which give up to 40MB/s, while IDE gives a maximum rate of 33 MB/s. The capacity of the disk also can lead to problems as the system can use unused disk capacity of a virtual memory capacity.

Obviously, applications that are more specific will be affected by other factors, such as:

- **Internet access.** Affected mainly by the network connection (especially if a modem is used).
- **CD-ROM access.** Affected by the interface to the CD-ROM.
- **Modelling software.** Affected by mathematical processing.
- **3D game playing.** Affected mainly by the graphics adaptor and graphics processing (and possibly the network connection, if playing over a network).

2.8 Practical PC system

At one time PCs were crammed full of microchips, wires and connectors. These days they tend to be based on just a few microchips, and contain very few interconnecting wires. The main reason for this is that much of the functionality of the PC has been integrated into several key devices. In the future, PCs may only require one or two devices to make them operate.

The architecture of the PC has changed over the past few years. It is now mainly based on the PCI bus. Figure 2.22 shows the architecture of a modern PC. The system controller is the real heart of the PC, as it transfers data to and from the processor to the rest of the system. Bridges are used to connect one type of bus to another. There are two main bridges: the system controller (the north bridge), and the bus bridge (the south bridge).

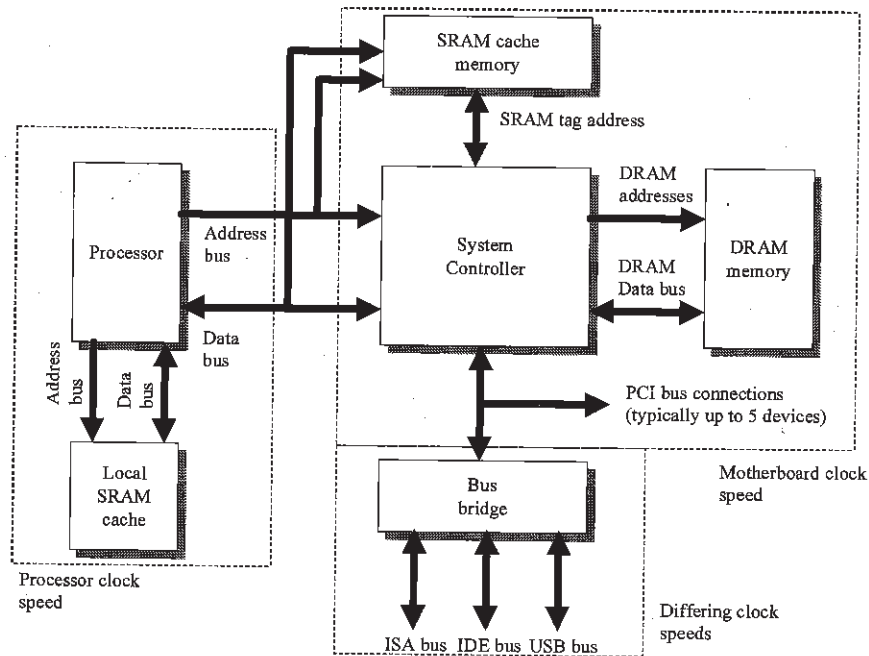


Figure 2.22 Local bus architecture

An example PC motherboard is illustrated in Figure 2.23. The main components are:

- **Processor.** The processor is typically a Pentium processor, which has a SEC (single-edge connector) or fits into a socket. The processor can run at a faster rate than the rest of the motherboard (called clock multiplication). Typically, the motherboard runs at 50MHz, and the clock rate is multiplied by a given factor, such as 500MHz (for a $\times 10$ clock multiplier).
- **System controller.** Controls the interface between the processor, memory and the PCI bus.
- **PCI/ISA/IDE Xcelerated Controller.** Controls the interface between the PCI bus and the ISA, USB and IDE busses.
- **I/O controller.** Controls the interface between the ISA and the other busses, such as the parallel bus, serial bus, floppy disk drive, keyboard, mouse, and infrared transmission.
- **DIMM sockets.** This connects to the main memory of the computer. Typically it uses either EDO DRAM and SDRAM (Synchronous DRAM). SDRAM transfers data faster than EDO DRAM as its uses the clock rate of the processor, rather than the clock rate of the motherboard.
- **Flash memory.** Used to store the program which starts the computer up (the boot process).
- **PCI connectors.** Used to connect to PCI-based interface adaptors, such as a network card, sound card, and so on.
- **ISA connectors.** Used to connect to ISA-based interface adaptors, such as a sound cards.
- **IDE connectors.** Used to connect to hard disks or CD-ROM drives. Up to two drives

can connect to each connector (IDE0 or IDE1) as a master or a slave. Thus, the PC can support up to four disk drives on the IDE bus.

- **TV out socket.** Used to provide an output which will interface to a TV, using either PAL (for the UK) or NSTC (for the US).
- **Level-2 cache (SRAM).** Used to store information from DRAM memory.
- **Video memory.** Used to store video information.
- **Graphics controller.** Used to control the graphics output.
- **Audio codec.** Used to process audio data.

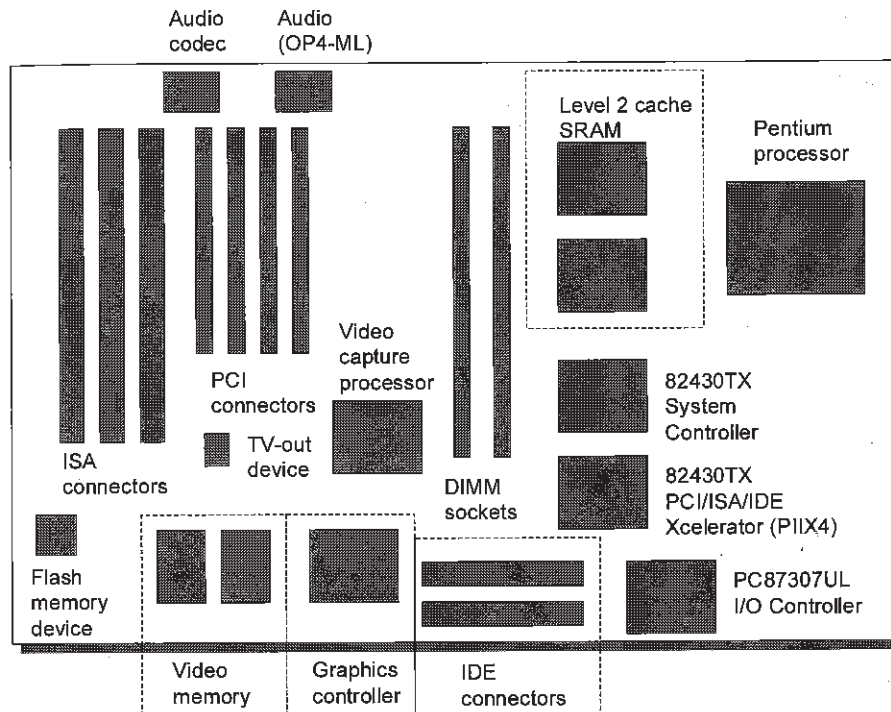


Figure 2.23 AN430TX board

2.5 Exercises

The following questions are multiple choice. Please select from a–d.

2.5.1 Which type of memory does not lose its contents when the power is withdrawn:

- | | |
|----------|----------|
| (a) ROM | (b) RAM |
| (c) DRAM | (d) SRAM |

2.5.2 Which type of memory uses a single capacitor and a transistor to store a single bit of data:

- (a) EPROM (b) ERAM
(c) DRAM (d) SRAM
- 2.5.3** Which type of memory requires its memory to be refreshed at regular intervals:
(a) EPROM (b) ERAM
(c) DRAM (d) SRAM
- 2.5.4** If a processor can operate on four bytes at a time, which is its classification:
(a) 8-bit (b) 16-bit
(a) 32-bit (b) 64-bit
- 2.5.5** Which of the following defines the amount of memory that can be accessed:
(a) Address bus (b) Control lines
(c) Handshaking lines (d) Data bus
- 2.5.6** Which of the following defines the number of bits that can be transmitted at a time:
(a) Address bus (b) Control lines
(c) Handshaking lines (d) Data bus
- 2.5.7** Which is the maximum data throughput for a 32-bit parallel data bus with a clocked data rate of 10MHz:
(a) 4MB/s (b) 40MB/s
(c) 32MB/s (d) 320MB/s
- 2.5.8** Which is the maximum data throughput for a serial bus which has a bit transmission time of 69.44 μ s:
(a) 6944 bps (b) 9600 bps
(c) 1440 bps (d) 14400 bps
- 2.5.9** How much memory can be accessed with a 20-bit address bus:
(a) 20B (b) 20KB
(c) 1MB (d) 20MB
- 2.5.10** How much memory can be accessed with a 32-bit address bus:
(a) 32B (b) 32KB
(c) 1GB (d) 32MB
- 2.5.11** Which interrupt does the primary serial port of a PC (COM1:) normally use:
(a) IRQ0 (b) IRQ3

- (c) IRQ4 (d) IRQ7
- 2.5.12** Which interrupt does the secondary serial port of a PC (COM2:) normally use:
 (a) IRQ0 (b) IRQ3
 (c) IRQ4 (d) IRQ7
- 2.5.13** Which interrupt does the system timer on the PC use:
 (a) IRQ0 (b) IRQ3
 (c) IRQ4 (d) IRQ7
- 2.5.14** Which interrupt was used to increase the amount of interrupts from 8 to 16:
 (a) IRQ0 (b) IRQ1
 (c) IRQ2 (d) IRQ15
- 2.5.15** Which interrupt is used by the keyboard:
 (a) IRQ0 (b) IRQ1
 (c) IRQ2 (d) IRQ15
- 2.5.16** What does ISR stand for:
 (a) Interval Status Register (b) Interrupt Status Register
 (c) Interrupt Service Routine (d) Interrupt Standard Routine
- 2.5.17** How is isolated memory differentiated from memory added I/O:
 (a) Different address bus (b) Different data bus
 (c) Control line differentiates between them (Memory/Isolated)
 (d) There is no differentiation as they are physically the same
- 2.5.18** How many addresses can be accessed in the address range 0000h to FFFFh:
 (a) 32 768 (32 kB) (b) 65 536 (64 kB)
 (c) 262 144 (256 kB) (d) 1 048 576 (1 MB)
- 2.5.19** How much physical memory can a DOS-compatible program access:
 (a) 32 768 (32 kB) (b) 65 536 (64 kB)
 (c) 262 144 (256 kB) (d) 1 048 576 (1 MB)
- 2.5.20** Which address is the interrupt control port register:
 (a) 0002h (b) 0020h
 (c) 0200h (d) 2000h
- 2.5.21** Which is normally the base address for the primary parallel port:
 (a) 0378h (b) 0278h
 (c) 03F8h (d) 02F8h

2.5.22 Contrast the operation of polling and interrupt-driven software when interfacing to external equipment.

2.5.23 Access a PC and determine the following:

Interrupt	Device connected
IRQ1	
IRQ3	
IRQ5	
IRQ7	
IRQ9	
IRQ11	
IRQ13	
IRQ15	
I/O address	Device connected
0060h, 0064h	
0070h	
0090h	
00F0h	
0278h	
02F8h	
0378h	
03F8h	
DMA channel	Device connected
DMA0	
DMA1	
DMA2	
DMA3	

2.6 Notes from the author

This chapter has introduced some of the key concepts used in defining computer systems. So, what is it that differentiates one PC system from another? It is difficult to say, but basically its all about how well bolted together systems are, how compatible the parts are with the loaded software, how they organise the peripherals, and so on. The big problem though is compatibility, and compatibility is all about peripherals looking the same, that is, having the same IRQ, the same I/O address, and so on.

The PC is an amazing device, and has allowed computers to move from technical specialists to, well, anyone. However, they are also one of the most annoying of pieces of technology of all time, in terms of their software, their operating system, and their hardware. If we bought a car and it failed at least a few times every day, we would take it back and demand another one. When that failed, we would demand our money back. Or, sorry I could go on forever here, imagine a toaster that failed half way through making a piece of toast, and we had to turn the power off, and restart it. We just wouldn't allow it.

So why does the PC lead such a privileged life. Well it's because it's so useful and multi-talented, although it doesn't really excel at much. Contrast a simple games computer against the PC and you find many lessons in how to make a computer easy-to-use, and to configure.

One of the main reasons for many of its problems is the compatibility with previous systems both in terms of hardware compatibility and software compatibility (and dodgy software, of course). The big change on the PC was the introduction of proper 32-bit software, Windows 95/NT.

In the future systems will be configured by the operating system, and not by the user. How many people understand what an IRQ is, what I/O addresses are, and so on. Maybe if the PC faced some proper competition it would become easy to use and become totally reliable. Then when they were switched on they would configure themselves automatically, and you could connect any device you wanted and it would understand how to configure (we're nearly there, but it's still not perfect). Then we would have a tool which could be used to improve creativity and you didn't need a degree in computer engineering to use one (in your dreams!). But, anyway, it's keeping a lot of technical people in a job, so, don't tell anyone our little secret. The Apple Macintosh was a classic example of a well-designed computer that was designed as a single unit. When initially released it started up with messages like I'm glad to be out of that bag and Hello, I am Macintosh. Never trust a computer you cannot lift.

So, apart from the IBM PC, what are the all-time best computers? A list by Byte in September 1995 stated the following:

1. MITS Altair8800	11. IBM AT
2. Apple II	12. Commodore Amiga 1000
3. Commodore PET	13. Compaq Deskpro 386
4. Radio Shack TRS-80	14. Apple Macintosh II
5. Osborne 1 Portable	15. Next Nextstation
6. Xerox Star	16. NEC UltraLite
7. IBM PC	17. Sun SparcStation 1
8. Compaq Portable	18. IBM RS/6000
9. Radio Shack TRS-80 Model 100	19. Apple Power Macintosh
10. Apple Macintosh	20. IBM ThinkPad 701C

And the Top 10 computer people as:

1. Dan Bricklin (VisiCalc)	11. Philippe Kahn (Turbo Pascal)
2. Bill Gates (Microsoft)	12. Mitch Kapor (Lotus 123)
3. Steve Jobs (Apple)	13. Donald Knuth (TEX)
4. Robert Noyce (Intel)	14. Thomas Kurtz
5. Dennis Ritchie (C Programming)	15. Drew Major (NetWare)
6. Marc Andreessen (Netscape Communications)	16. Robert Metcalfe (Ethernet)
7. Bill Atkinson (Apple Mac GUI)	17. Bjarne Stroustrup (C++)
8. Tim Berners-Lee (CERN)	18. John Warnock (Adobe)
9. Doug Engelbart (Mouse/Windows/etc)	19. Niklaus Wirth (Pascal)
10. Grace Murray Hopper (COBOL)	20. Steve Wozniak (Apple)

One of the classic comments of all time was by Ken Olson at DEC, who stated, that there is no reason anyone would want a computer in their home. This seems farcical now, but at the time, in the 1970s, there were no CD-ROMs, no microwave ovens, no automated cash dispensers, and no Internet. Few people predicted them, so, predicting the PC was also difficult. But the two best comments were:

Computers in the future may weigh no more than 1.5 tons. Popular Mechanics.
I think there is a world market for maybe five computers, Thomas Watson, chairman of IBM, 1943.

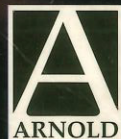
As more and more equipment is interface driven, either by the use of controllers or directly from PCs, the question of which bus to use is increasingly important. *Computer Busses* has been designed to help choose the best type of bus for the particular application.

The text analyses busses from a top-down perspective and draws examples and applications from real busses with plenty of case studies.

-  Provides a complete guide to computer busses
-  Contains application-specific programme examples
-  Plenty of real-life case studies
-  Covers local, instrumentation and network busses, as well as bus programming and protocols

This book is essential reading for students of software engineering and electronic design, and for disciplines from production engineering to process control, during project work. It will also be a handy reference book for professional engineers, system designers, consultants and technical support.

www.arnoldpublishers.com



Copublished in North America by
CRC Press LLC, Boca Raton, FL
CRC ISBN 0 8493 0825 9



0-340-74076-0

0825

BUCHANAN

COMPUTER BUSES

ENGIN
TK
7895
.B87B83
2000