

CERN 85-09
9 July 1985

ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE
CERN EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

1984 CERN SCHOOL OF COMPUTING

Aiguablava, Catalonia, Spain
9-22 September 1984

PROCEEDINGS

GENEVA
1985

© Copyright CERN, Genève, 1985

Propriété littéraire et scientifique réservée pour tous les pays du monde. Ce document ne peut être reproduit ou traduit en tout ou en partie sans l'autorisation écrite du Directeur général du CERN, titulaire du droit d'auteur. Dans les cas appropriés, et s'il s'agit d'utiliser le document à des fins non commerciales, cette autorisation sera volontiers accordée.

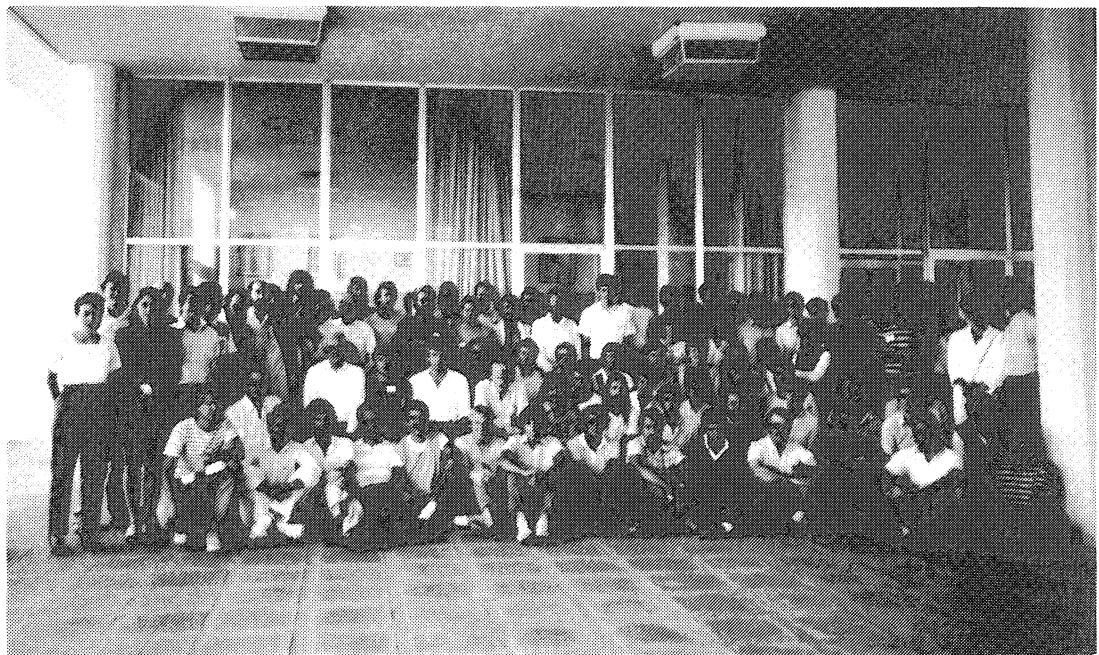
Le CERN ne revendique pas la propriété des inventions brevetables et dessins ou modèles susceptibles de dépôt qui pourraient être décrits dans le présent document; ceux-ci peuvent être librement utilisés par les instituts de recherche, les industriels et autres intéressés. Cependant, le CERN se réserve le droit de s'opposer à toute revendication qu'un usager pourrait faire de la propriété scientifique ou industrielle de toute invention et tout dessin ou modèle décrits dans le présent document.

Literary and scientific copyrights reserved in all countries of the world. This report, or any part of it, may not be reprinted or translated without written permission of the copyright holder, the Director-General of CERN. However, permission will be freely granted for appropriate non-commercial use.

If any patentable invention or registrable design is described in the report, CERN makes no claim to property rights in it but offers it for the free use of research institutions, manufacturers and others. CERN, however, may oppose any attempt by a user to claim any proprietary or patent rights in such inventions or designs as may be described in the present document.

ABSTRACT

The eighth CERN School of Computing covered subjects mainly related to computing for elementary-particle physics. These Proceedings contain written versions of most of the lectures delivered at the School. Notes on the following topics are included: trigger and data-acquisition plans for the LEP experiments; unfolding methods in high-energy physics experiments; Monte Carlo techniques; relational data bases; data networks and open systems; the Newcastle connection; portable operating systems; expert systems; microprocessors -- from basic chips to complete systems; algorithms for parallel computers; trends in supercomputers and computational physics; supercomputing and related national projects in Japan; application of VLSI in high-energy physics, and single-user systems.



PREFACE

From 9-22 September 1984 the CERN School of Computing was held in Aiguablava, Catalonia, Spain. Eighty-six students participated in the School; the majority came from high-energy physics institutes in the Member States of CERN. The School marked the re-accession of Spain as a CERN Member State. This country also provided a large contingent of participants.

The lecture programme covered topics in computing which attract much interest, such as expert systems, relational data bases, supercomputers and their use in computational physics. Other subjects presented at the School were closely related to computing for particle physics. Data acquisition and filtering for LEP experiments, data reduction and Monte Carlo techniques, open systems architecture, operating systems, and microprocessors were the subject of lectures in this field. Other topics covered were chip design (VLSI and special circuits), data storage techniques, and the Newcastle connection. All lectures invariably attracted a high attendance.

The contributions of all lecturers were of a high standard and their efforts largely account for the success of the School. The lectures were given in an informal atmosphere, in which good contacts between lecturers and students were established. We wish to express our gratitude to all the lecturers for their invaluable contributions.

This School, the eighth in the series, innovated by introducing a tutorial with hands-on experience. A VAX 11/780 and 16 terminals were installed and kept running daily until midnight. This facility was very popular with the students and we are indebted to Dietrich Wiegandt, who ran the tutorial and the exercises with great enthusiasm. The assistance of R. Merat from the University of Geneva in bringing up the UNIX system on the VAX 11/780 is gratefully acknowledged. We are indebted to the Digital Equipment Corporation, Barcelona, Spain, who put the VAX at our disposal, free of charge and provided technical assistance for its installation. Likewise we want to thank the Xerox Company, Barcelona, for putting at our disposal, also free of charge, a photocopier.

The local organizers did an outstanding job in arranging for all the facilities, as well as for the very interesting excursions to the Dali Museum in Figueras, Ampurias, to the monastery of Sant Pere de Rodes, and the guided visit to Gerona and Pals. The many Spanish people who arranged for all this are thanked very warmly. Especially Dr. F.J. Armada-Comyn and Professor Marti Verges are congratulated for their excellent work and for solving all problems with a smile. We are very grateful to Dr. J. Rubio for his initiative and his continual support. The generous financial contribution of the Junta de Energia Nuclear, Madrid, is very gratefully acknowledged.

The site of the School, the Parador "Costa Brava" at Aiguablava, also offered everything to contribute to the success of the School. The manager of the Parador, Mr. Gomi, his staff and particularly 'le chef' are warmly thanked for their efficient and kind services.

The efforts of Mrs. Edith Lafouge of the Scientific Reports Editing Section in preparing these Proceedings for printing are greatly appreciated.

Last, but not least, we thank the secretarial staff for their assistance, in particular Mrs. Ingrid Barnett for her efficient work before, during, and after the School.

C. Verkerk
Editor

* * *

ADVISORY COMMITTEE

ARMADA, J. JEN, Madrid, Spain
BLOBEL, V. Hamburg University, Fed. Rep. of Germany
BOLLIET, L. Laboratoire IMAG, Grenoble, France
CHURCHHOUSE, R.F. University College, Cardiff, U.K.
DOBINSON, R.W. University of Illinois at Champaign-Urbana, U.S.A.
and CERN, Geneva, Switzerland
GABATHULER, E. Liverpool University, U.K.
LEVRAT, B. Geneva University, Switzerland (Chairman)
RAMOS, I. Valencia University, Spain
RUBIO, J. JEN, Madrid, Spain
VERGES, M. Universidad Politecnica de Barcelona, Spain
VERKERK, C. CERN, Geneva, Switzerland
ZANELLA, P. CERN, Geneva, Switzerland
BARNETT, I. CERN, Geneva Switzerland (Organizing Secretary)

CONTENTS

	<u>Page</u>
PREFACE	v
WELCOME ADDRESS <i>B. Levrat</i>	1
TRIGGER AND DATA-ACQUISITION PLANS FOR THE LEP EXPERIMENTS <i>W. von Rilden</i>	4
DATA NETWORKS AND OPEN SYSTEMS <i>R.A. Rosner</i>	28
THE NEWCASTLE CONNECTION: A SOFTWARE SUBSYSTEM FOR CONSTRUCTING DISTRIBUTED UNIX SYSTEMS <i>B. Randell</i>	74
UNFOLDING METHODS IN HIGH-ENERGY PHYSICS EXPERIMENTS <i>V. Blobel</i>	88
MONTE CARLO TECHNIQUES <i>J. Salicio</i>	128
EXPERT SYSTEMS: AN OVERVIEW <i>F. Verdejo</i>	161
ALGORITHMS FOR PARALLEL COMPUTERS <i>R.F. Churchhouse</i>	170
TRENDS IN SUPERCOMPUTERS AND COMPUTATIONAL PHYSICS <i>T. Bloch</i>	192
SUPERCOMPUTING AND RELATED NATIONAL PROJECTS IN JAPAN <i>K. Miura</i>	202
APPLICATION OF VLSI IN HIGH-ENERGY PHYSICS <i>B.D. Hyams</i>	223
MICROPROCESSORS: FROM BASIC CHIPS TO COMPLETE SYSTEMS <i>R.W. Dobinson</i>	237
FORMAL ASPECTS IN DATABASES <i>C. Delobel</i>	306
PORTABLE COMPUTERS - PORTABLE OPERATING SYSTEMS <i>D. Wiegandt</i>	338
SINGLE USER SYSTEMS <i>I. Willers</i>	362
List of Participants	374

WELCOME ADDRESS

B. Levrat

University of Geneva, Switzerland

It is a great pleasure to welcome you to the 1984 CERN School of Computing. I hope that the fact that the buses which should have brought you straight to this place took you instead through the twisting roads of the Costa Brava until late at night will not be held against the School organizers. It is a bit hard for bus drivers to distinguish CERN School participants from tourists! But I should not joke about it and I beg those who were inconvenienced by the long bus ride to accept our apologies. When we chose Aiguablava, we were aware that it was a bit difficult to reach. We felt, however, that the wonderful setting and the superb conditions under which we will work and relax were worth the effort. Now that you have seen the place in full daylight, I am sure that you will agree with us.

First of all, let me introduce the personalities who are going to say a few words at this opening ceremony and whose continued support has made the present School possible: Professor J. Rubio is Scientific Director of the Junta de Energia Nuclear in Spain; Professor Manuel Martí i Recober, Director General D'Ensenyament Universitari de la Generalitat de Catalunya; and Professor H. Schopper, Director-General of CERN.

It is also a pleasure to greet a number of very distinguished visitors and thank them for their interest in the School. Professor A. Sierra is Rector of the University of Barcelona. Professor S. Ting, Nobel Prize winner, is actively engaged in collaborations with CERN and with high-energy physics groups in Spain. He is here with Dr. W.A. Wallenmeyer, Leader of the Division of High-Energy Physics of the United States Department of Energy, Dr. B. Hildebrand of the same Department, and Dr. J.A. Ruiz Lopez-Rùa, Director for International Relations of the Spanish Department of Energy. Professor Pascual is President of the Spanish High-Energy Physics Plan.

I must also thank Professor C. Rubbia, who accepted to be our keynote speaker in spite of his many other commitments. Let me also mention four key people who devoted much of their time to preparing the School and who must feel elated to see that everything is going well. They are Professor M. Vergès from Barcelona, who is also co-director of the School, Dr. J. Armada from Madrid, and Mr. C. Verkerk and Mrs. I. Barnett, both from CERN.

The first CERN School of Computing took place in Varenna, Italy, in 1970, and I was a member of the first Advisory Committee as well as a lecturer. When I look back 14 years, I am impressed by the magnitude of the changes that CERN and computing have gone through.

CERN revolved around one main accelerator, the Proton Synchrotron, and the experimental groups had an average of 10 members, except in bubble chamber experiments. All the physicists knew one another and group meetings were held in the group leader's office.

Computers were very expensive and their use was optimized. All programs were run in 'batch mode' and interactive computing was barely emerging from 'on-line' experiments and bubble-chamber picture analysis.

Today, computers are more powerful, cheaper, interactive, and try desperately to qualify as friendly. Computers do not stand alone in their aseptic computer rooms. They are the

hub of communication networks which make it easy to access the entire world from the terminal in one's office. CERN is moving at a fast pace towards LEP and the L in LEP means LARGE: Large experiments conducted and analysed by large groups. The problems of today are problems of communication among people, among computers, and among systems; they are not restricted to the scientific community but are deeply changing the fabric of society.

One thing remains however. High-energy physics is playing a pioneering role in harnessing new technologies. What is done in your laboratories will influence the whole scientific community. This is particularly true of CERN, which is setting the pace for Europe. It may not be felt from the inside and may not be correctly perceived from the outside, but the thousands of scientists who come into contact with the fine computing facilities available in Meyrin before going back to their jobs in universities, government offices, or industry, are spreading what they learn at CERN. It is to be hoped that they are impressed not only by the most modern hardware technology but that they also give proper attention to the underlying software and design which delivers its power to users.

One of the reasons for the CERN School of Computing is to teach about proven techniques for achieving very ambitious experiments, digesting billions of bits of data, simulating complex environments, and solving difficult theoretical problems. New techniques and their impact on the current limitations of the above-mentioned problems will also be thoroughly explored. Advances are also taking place in programming, but a breakthrough in hardware will hit the marketplace within 2 or 3 years, while advances in software will often take up to 10 years to be recognized and widely used. It is amusing to note that hardware developments are always recognized as progress, while new software techniques always give rise to spirited controversies.

The role of the Advisory Committee is to prepare a balanced programme which will deal equally with the hunger of high-energy physicists for massive computing power, the new techniques in networking, operating systems and data bases, and entirely new developments in software leading to expert systems and, maybe, to a knowledge-based society if 'fifth generation' computers live up to what Japan has announced. The true results of our efforts appear in the School brochure and are reflected in the Proceedings, but let us have a quick look at the global picture.

Data acquisition and filtering as well as data reduction receive the attention they deserve in the lectures by Dr. W. von Rueden from CERN and by Professor V. Blobel from the University of Hamburg and DESY. Large-scale computing is also required at CERN from Monte Carlo calculations (presented by Dr. J. Salicio from the JEN, now at DESY) and some modern work in theoretical physics. We had Ken Wilson lined up for this subject but he had to cancel his visit because of other commitments. Instead, we called on Dr. T. Bloch, who is in charge of the CRAY-1 computer at the Ecole Polytechnique in Paris. Dr. Bloch will prepare the stage for a few speakers from industry who will talk about future developments in supercomputers (Dr. K. Miura from Fujitsu), data storage technology (Dr. M.D. Canon from IBM), and VLSI (Dr. F.P. Carrubba from Hewlett Packard). It is the first time that a CERN School of Computing will hear from designers so closely connected with the shaping of future technologies.

The help that these new technologies can bring to an experimentalist will be looked at by Dr. B. Hyams, who is Leader of the EP Division at CERN, while Professor R.F. Churchhouse will show that much remains to be done in the domain of software and algorithms for parallel machines. Also connected with technological advances, Dr. R.W. Dobinson, presently at the University of Illinois, will talk about use and abuses of microprocessors. One important development brought about by micro-electronics is the ubiquitous personal computer, which will be the subject of the presentation by Dr. I. Willers who will touch on social implications.

Computers can be connected through networks. Dr. R.A. Rosner from the University of London Computer Centre will show that open system models for interconnection can bring computers from different manufacturers to talk to each other. Some operating systems will greatly facilitate the writing of communication protocols. It is the case with UNIX and Professor B. Randell who built the 'Newcastle connection' will explain how.

UNIX plays such an important role in the research community that the Advisory Committee thought that, in addition to a series of lectures on the subject given by Dr. D. Wiegandt from CERN, some hands on experience would be very desirable. A laboratory has been set up next door, thanks to substantial efforts by Professor M. Vergès and his assistants, the JEN, DEC, CERN, and the University of Geneva. The laboratory and the tutorials prepared by Dr. Wiegandt are for first-time users. Experts can lend a friendly hand, go to the beach, or use the computer for their own purposes.

So much data accumulate at CERN that there is a great demand for Data Base Management Systems to be applied to particle data, cable layouts, payroll, conference scheduling, etc. Professor C. Delobel, from the University of Grenoble, will explain how the relational formalism makes the specifications and querying much easier. Dr. S. Santiago will talk about the main data bases in existence at CERN today.

Advance software techniques will be treated by two distinguished speakers from Spain. Professor M. Verdejo from the University of Donostia will present an overview of the current developments of expert systems, while Professor M. Vergès will talk about logic programming with some examples in Prolog that you can program on the VAX next door.

The Advisory Committee is well aware that it left out many important subjects. It feels, however, that there is enough to make the coming two weeks very interesting provided that you cooperate. We have brought together excellent teachers and gifted students. We hope they will interact not only in the lecture room but also during breaks, drinks, meals, and leisure.

Work will start in half an hour when our keynote speaker, Professor Carlo Rubbia from CERN, will show how hardware, software, and good physics ideas can be put together to produce some truly magnificent experiments.

TRIGGER AND DATA-ACQUISITION PLANS FOR THE LEP EXPERIMENTS

Wolfgang von Rüden
CERN, Geneva, Switzerland

Abstract

Four experiments have been accepted for phase 1 of LEP: ALEPH, OPAL, L3, and DELPHI. These experiments have each of the order of 100,000 or more electronic channels producing more than 100 Mbytes of raw data per second. Extensive zero-suppression, data reduction and formating as well as intelligent triggers are needed to reduce the amount of data to be written to tape to 0.1-1 Mbyte per second.

As the various detector components are developed in laboratories distributed all over Europe, in the US and in China and Japan, good communications are essential during the development phase and later for distributed data analysis.

We give a brief overview on the LEP accelerator and introduce the four detectors. We discuss the trigger and the data acquisition for the four experiments and summarize the different techniques used.

1. HISTORY

During the years 1976 to 1979 preliminary discussions started in the European High-Energy Physics community about a follow-up programme for the CERN Super Proton Synchrotron (SPS). Four alternatives were proposed, namely a 10 TeV Proton Synchrotron, a 400 x 400 GeV pp collider, an ep collider, and a large e^+e^- ring. The last option met with the greatest interest and the project was named LEP, the Large Electron-Positron Collider.

In 1979 preliminary studies started in Les Houches, followed by the Uppsala conference in 1980 and the conference in Villars in 1981. The CERN Member States approved the project unanimously in December 1981. At that time collaborations had already been formed, and in January 1982 seven Letters of Intent were submitted to the LEP Committee (LEPC) whose task was to select four experiments. The first session took place in March 1982, where questions arose concerning the feasibility of the projects. Referees were appointed, and in summer 1982 the following experiments were accepted:

L1 ALEPH
L2 OPAL
L3
L4 DELPHI.

All four collaborations submitted detailed Technical Reports in April 1983. In addition, special reports on trigger and data handling were sent to the LEPC in January 1984.

The present talk is based on the Technical Reports [1-4], the reports on data handling [5-8] and on private communications from the collaborations. The reader should keep in mind that we present the plans of the experiments and that some of the information contained in this presentation will be obsolete by the time this report is published.

2. THE LEP MACHINE

As the topic of the present talk is data acquisition, only a very small section can be devoted to the superb LEP accelerator project. Fig. 1 shows an aerial view of the Geneva area with the airport at the bottom of the picture and the Jura mountains in the back. The present CERN accelerators, the PS, the ISR and the SPS, can be seen as well as the planned implementation of LEP, which has a circumference of nearly 27 km (!). The locations of the eight access points are marked in fig. 2. Only the four even-numbered interaction regions will be equipped with experiments during the first phase of LEP; pit 2 has been assigned to the L3 collaboration, pit 6 to OPAL, whilst the assignment for ALEPH and DELPHI is still under discussion.

Figure 3 shows a vertical cut with the Jura on the left-hand side and the lake of Geneva on the right. Most of the LEP tunnel will be in the "molasse", a convenient material for tunnelling. Only a small part, which is under the Jura, consists of limestone and may therefore give some unexpected problems. The ring is inclined by 1.4% and the depth varies between 50 m and 150 m.

The layout of an interaction region is shown in fig. 4. The "straight" pipe will receive the accelerator. The tunnel on the left is for machine equipment, mainly the klystrons for the RF. The experimental hall has a diameter of about 15 m and a length of 70 m. Three access shafts are provided: the leftmost for machine access, the large one in the centre to bring the experiment down, and the small shaft on the right will be used by the experimentalists. Two 40-ton cranes will be installed in the experimental hall to assemble the detector. The detectors (except L3) will be movable from the beam position to the "garage" position for maintenance work. The surface buildings provide storage room for equipment, cranes to bring it down and also space for control rooms, workshops, some offices, etc. Some of the buildings are devoted to machine equipment.

The interaction region number 2, for the L3 collaboration, differs from the others as it is oriented parallel to the machine tunnel. Also, the L3 detector has too much weight to be moved, so it will be assembled directly into its final position.

Table 1 gives a few machine parameters. The particles will make 1 turn every 88 μ s and as there are 4 bunches, the beam crossing will occur every 22 μ s at each interaction point. According to the present planning, and if everything goes well, the beam for the experiments is expected for the end of 1988.

3. THE FOUR LEP DETECTORS

We give a short description at the four detectors to make the understanding of the trigger and data-acquisition easier. For details, the interested reader should refer to the Technical Reports issued by the collaborations [1-4].

The scenario for the detector construction is very similar for the four collaborations. The various pieces of the detectors will be constructed in institutes spread all over Europe and even to the United States, China and Japan. Therefore, all experiments are confronted with communications problems, which are in particular delicate for the on-line system.

3.1 The ALEPH detector

The ALEPH collaboration, spokesman J. Steinberger, is formed by 25 institutes from 8 countries and counts today some 330 physicists and engineers. The detector [1] is shown in fig. 5. Looking from the beam-pipe outwards, it is composed of an inner trigger chamber (4), the time projection chamber (5), the electromagnetic calorimeter (6), the superconducting coil (7), the hadron calorimeter (8) and finally the muon detector (9). A luminosity monitor (3) surrounds the beam-pipe at each end of the TPC.

3.2 The OPAL detector

The OPAL collaboration, spokesman A. Michelini, is formed by 180 physicists and engineers, coming from 19 institutes in 9 countries. The detector [2] design is more conservative, and the components are based on known techniques, of course on a much larger scale than built up to now.

Starting from the beam-pipe (fig. 6) we recognize the central detector (vertex chambers and JET chamber), surrounded by the solenoid, followed by time-of-flight counters, the electromagnetic and the hadron calorimeters, and the muon chambers.

The simulation of a two-jet event in this detector is shown in figs. 7 and 8 [9].

3.3 The L3 detector

This collaboration, spokesman S.C.C. Ting, is made up of 36 institutes from 12 countries and counts about 360 physicists and engineers. The arrangement of the L3 detector [3] in its experimental hall is shown in fig. 9. From the inside to the outside, we find first a time expansion chamber, followed by electromagnetic shower counters made up of 12,000 BGO crystals. Thereafter come the hadron calorimeter and the muon drift chambers. All these components are surrounded by a very large magnet.

3.4 The DELPHI detector

There are 34 institutes from 17 countries, with 280 physicists and engineers, forming the DELPHI collaboration, spokesman U. Amaldi. The DELPHI detector [4] has the

largest variety of detector elements using different techniques. Starting from the beam-pipe (fig. 10), there are a vertex chamber, an inner detector, a time projection chamber, the barrel ring imaging Cerenkov (RICH), the outer detector and the barrel electromagnetic calorimeter. These components are surrounded by a superconducting coil, after which follow a scintillator hodoscope, the hadron calorimeter, and two layers of muon chambers. Going from the centre to the end-plates, we find after the TPC a set of three forward chambers (A,B,C) interspersed by a liquid RICH and a gas RICH, then the electromagnetic calorimeter and finally the end cap hadron calorimeter.

4. THE TRIGGER SYSTEMS

Because of the different properties of the four detectors, the trigger designs also vary in complexity, and response time. The information given here is based on the reports on trigger and data acquisition of January 1984 [5-8]. For more recent information, the on-line coordinators can be contacted:

ALEPH	W. von Rüden, CERN,
OPAL	H. von der Schmitt, Bonn and CERN,
L3	M. Fukushima, DESY,
DELPHI	J. Allaby, CERN.

In the following, we present the basic principles of the different triggers. For the timing and rates, Table 2 provides a comparison of the four detectors.

4.1 The ALEPH trigger

The aim is to trigger on all physics events, whilst rejecting background as much as possible. Three independent conditions have been defined:

1. ≥ 2 minimum ionizing tracks;
2. ≥ 1 minimum ionizing track and one energy cluster (low threshold);
3. total electromagnetic or hadronic energy above a (high) threshold.

The trigger is performed at three levels: level 1 must respond within 1.5 μ s (including cable delays) of bunch crossing, to open the gate of the TPC. This time corresponds to ~ 7.5 cm drift space being lost at the end of the tracks, for those passing through the end-plates. The detector components contributing to this level of triggering are:

- the inner trigger chamber,
- the electromagnetic calorimeter,
- the hadron calorimeter,
- the muon chamber,
- the luminosity monitor.

Figure 11 shows a block diagram of the ALEPH level-1 trigger logic. The 72 segments for the calorimeters are formed by grouping towers of the calorimeter into "super" towers.

After the drift time of the TPC ($\sim 40 \mu s$), there is additional information about the tracks in the TPC pointing towards the vertex. This information is obtained by a special hardware processor during the TPC drift time using special trigger pads; therefore, no readout is needed to process this information, which is used to generate the second-level trigger. If the decision is positive the readout of the various sub-detector components is started.

The last stage is the event processor, a computer capable of running the reconstruction programme, which will see for the first time a complete event. Its task would be to reject further background events and/or clarify events for the off-line analysis.

A trigger supervisor will assure the proper gating and distribution of the trigger signals to the different sub-detectors; it will also be used to perform consistency checks and to guarantee the proper response of all needed components.

4.2 The OPAL trigger

The OPAL detector will trigger on

1. hadronic jets,
2. charged-lepton pairs,
3. radiative production of $Z^0 \rightarrow \nu$,

and also on two-photon processes and free quarks. As the decision time can be as long as $18 \mu s$ (4 are needed to clear for the next bunch crossing), a very sophisticated trigger, including all detector components, can be made. Indeed, information comes from

1. the electromagnetic and hadron calorimeters: energy and topological information;
2. the tracking chambers: vertex, jet and z-chambers;
3. the muon chambers;
4. the forward detector.

This trigger is supposed to bring the 50 kHz bunch crossing rate down to ≤ 4 Hz!

4.3 The L3 trigger

For the L3 detector, six different triggers are planned with the following properties:

1. Muon trigger: ≥ 1 track pointing to the vertex, $p_T \geq 2$ GeV/c.

2. Energy trigger: the sum of the electromagnetic shower counters (BGO) and of the hadron calorimeter exceeds 20% of the centre-of-mass energy.
3. Charged-particle trigger: ≥ 2 tracks pointing to the vertex, $p_T \geq 0.3$ GeV/c. If this trigger is too loose, ask for some energy deposit in the calorimeters.
4. Single-photon trigger: defined by an isolated energy deposit in the BGO crystals.
5. Small-angle trigger: requires some energy deposit in at least one side of the small angle calorimeter and a charged track or energy in the central calorimeter.
6. Luminosity trigger: energy deposit in both sides of the detector.

Figure 12 gives details of the level-1 and level-2 triggers. Note, that 4,500 input signals are used to construct the trigger, which might give an idea of the complexity of such a device. The timing and the rates for the different stages are represented in fig. 13.

4.4 The DELPHI trigger

The DELPHI trigger system is based on four levels. For the first level (~ 2 μ s) the requirements are

1. ≥ 1 charged track of ≥ 2 GeV/c momentum.
2. An energy deposit in the calorimeters for neutral events.
3. Two coincident track elements from the inner detector, outer detector, TPC end-plates and forward chambers (depending on the detector region).

If these requirements are too loose (rate above 1 kHz), information from the electromagnetic calorimeter, the time-of-flight counters, or the towers of the hadron calorimeter can be used in addition.

At the second level (~ 35 μ s) more detector components contribute to the trigger decision:

1. the TPC: - a track points to the vertex;
- cut on p_T ;
2. the HPC: correlate tracks and energy clusters;
3. the muon chambers;
3. the electromagnetic and the hadron calorimeter for neutral hadrons.

Level 3 requires information from the readout system. Each detector will have a microprocessor (GPM [10]) to perform a selective readout. The combined information will be transmitted to the main level-3 processor, likely to be a XOP [11].

Finally, level 4 will use emulators for further filtering of events. Figure 14 gives an overview of the DELPHI trigger system.

4.5 Trigger summary

As each collaboration applies different terms for similar items, and as the reports on trigger and data acquisition are organized in different ways, a comparison is difficult. Table 2 is an attempt to bring the information together.

The different "levels" defined there do not necessarily coincide with the naming found in the reports: Level 1 includes actions taking place between beam crossings (22 μ s). There is a large difference in the expected rates between OPAL (4 Hz) and DELPHI (1000 Hz). Slower detectors such as a TPC will contribute to decisions in level 2. Here the rates of the four detectors are about the same (4-20 Hz). Level 3 (if used) reduces the rates by another order of magnitude.

Finally, the event processors will really see the "full" event and might take decisions based on sophisticated algorithms. In general, emulators or banks of microprocessors are supposed to deliver massive CPU power for this job, at a reasonable cost. The table also shows the expected event size and deduces from that the number of 6250 bpi tapes written per day. Note that for an expected running time of 100 days, each collaboration will produce about 10,000 tapes per year!

5. THE READOUT SYSTEMS

Despite similar requirements, the readout systems of the four detectors have adopted quite different solutions. ALEPH, L3, and DELPHI have chosen FASTBUS as their data-acquisition standard; OPAL will use FASTBUS only at the level of the front-end electronics and base the readout and the processor modules on the VME/VMX bus standard. Using FASTBUS does not mean that the other three experiments would look alike. The flexibility of FASTBUS allows the user to dream up virtually any configuration.

The location and interconnection of the mini-computers is also different for the four experiments, but at least all have decided to use the VAX family of computers running VMS. Ethernet seems to emerge as the preferred solution for the local area network (LAN).

5.1 The ALEPH readout

Figure 15 shows a typical ALEPH sub-detector arrangement during the development phase. The equipment or sub-detector computer is used to develop and understand the detector and to prepare the tools for the monitoring and calibration. The requirements vary from one sub-detector to another; for example, the TPC is the most demanding part,

producing ~ 80% of the data per event. Its electronics will fill more than 100 FASTBUS crates. The connection via the LAN to the international network allows close contact between the sub-detectors. In fact, such a sub-detector set-up corresponds in its complexity to today's large experiments.

The connection to the central readout can be seen from fig. 16. During normal data taking the readout is controlled centrally, while the "local" VAX can "spy" on events from its sub-detector. The function of the event builder (EB) is to collect the pieces of an event from the different readout controllers, which are the masters in the front-end crates, and to store the combined information in its memory. Whenever a sub-detector computer needs data, it might request a copy of the next event from the EB. The main data stream will not be affected by the speed at which a local computer can absorb data.

The overall layout (preliminary) is sketched in fig. 17. The sub-detector computers and the main on-line computer are located in the surface building, at ~ 150 m from the detector. Optical-fibre links are required for the long-distance FASTBUS connection. This solution has the advantage that all computers can be connected via Digital's "Computer Interconnect CI", a high-speed serial link supporting shared disks and tapes via the "hierarchical storage controller HSC50". Also, this link will serve as a fast communication channel between the sub-detector computers and between these and the large VAX. The arrangement in one room gives the additional benefit of a clean "computer floor" and eases the maintenance.

The Ethernet serves to connect terminals via concentrators, to provide a printer/plotter station, and a link to the "slow control" equipment (see later). It will extend down to the counting rooms, where terminals and printer are needed near the electronics. It should also provide the link to the CERN-wide network and through it the access to the international connections.

5.2 The OPAL readout

Figure 18 shows the latest version of the OPAL VME/VMX readout system and the connection to CAMAC. As this information was received just before the Symposium, the reader is asked to contact the OPAL collaboration for more information.

5.3 The L3 readout

Depending on the particular sub-detector, the front-end electronics is placed either on the detector itself and interfaced to FASTBUS, or it is directly built in FASTBUS. Taking the muon chambers as an example, L3 will use the LeCroy 1800 system with the 96-channel TDCs (model 1879) and the readout controller 1821. The overall control is done by a VAX 11/780. For the hadron calorimeter, the ADCs are mounted directly on the detector and a private databus will end in a multiplexor interfaced to FASTBUS.

The BGO readout deserves special attention, as each of the 12,000 crystals will have its own microcomputer (M146805) for readout and control. They are connected in 150 groups of 80, controlled by 150 micros; these are, in turn, organized as 8 groups of about 20. The 8 controllers can be either CAMAC or FASTBUS modules, controlled by a machine with a CPU power equivalent to a small VAX.

Without going into the details of every sub-detector, it can be said that L3 will have, inside each sub-detector, several parallel data streams merging into a buffer memory, managed by an EB (fig. 19). The assembled sub-events are passed through a central EB to the emulators memory; from there they can be transferred (after treatment) to the host computer (VAX 11/790) and/or to an extra emulator for full event reconstruction; they are then further analysed by another VAX 11/790.

5.4 The DELPHI readout

DELPHI stresses the modularity of the data-acquisition system. Each major sub-detector component will have its own "equipment computer"; all these are interconnected via a LAN (fig. 20). There will be a "OPS" computer, a "large general-purpose computer facility serving the whole experiment", and a "DAC" computer whose task is the transfer of data from the main data acquisition to the tape. Extra computing power for both machines is foreseen in the form of emulators on a separate dedicated LAN. The general LAN will provide the connections to the CERN-wide and European-wide networks.

5.5 Network connections

Good communications between the various home institutes and CERN are vital for all LEP collaborations. At the moment, the network is growing all the time. Most of the machines at CERN are already connected via DECnet, which extends, via the INFN gateway, to Italy, where DECnet has been in use for several years already. For France, Switzerland and Germany, public X.25 networks are proposed. Great Britain has its own scientific network JANET, based on the "coloured book" software and X.25. The network extends to CERN and it might even be possible to run DECnet over JANET to have a uniform connection. The GIFT project at CERN is supposed to provide gateways between incompatible networks.

In the final layout there will be several levels of networking: local connections at the experiment, regional on the CERN site, and the European or world-wide. The example in fig. 21 shows how DELPHI sees the networking, which is representative of the other experiments systems.

6. SLOW CONTROL

This includes monitoring of voltages, currents, temperatures, regulation of power supplies, control of gas systems, monitoring of interlocks, etc. The intention is to use the equipment developed by the LEP control group for the accelerator. It is based on the M6809 microprocessor, the form-factor is a single-height Eurocard using the G64 bus. FLEX has been selected as the operating system. To connect multiple crates, two types of LAN are envisaged: UTI-net [12] or the MIL-1553 standard.

For more demanding tasks, VME-based M68000-type systems are likely to be used. The connection to the main data-acquisition system could go via VME-based Ethernet controllers or via direct interfaces to the VAX.

7. SUMMARY OF PROCESSORS

For the first- and second-level triggers, special hardware machines are proposed, based on look-up tables, coincidence matrices and similar. For the higher-level triggers, some experiments propose to use fast bit-slice processors such as KOP [11] or M68000-type machines in the FASTBUS standard.

For data reduction and readout at the crate level, the M68000 is the favourite, either in FASTBUS or in VME (OPAL). L3 has a special solution for the BGO readout using one CMOS microprocessor per channel (12,000 units).

At the level of the event processor or event filter three solutions are competing: emulators such as the 370E [13] or the 3081E [14], arrays of microprocessors (Fermilab ACP project) [15] or a set of microVAXes. The last one is very attractive because of software compatibility with the general system. However, the CPU power per dollar might exclude this solution.

All four LEP experiments have decided to use VAX computers as the minis for the data acquisition and monitoring.

For slow control applications the plans are to use the M6809 based on G64 or the M68000 in VME.

Last but not least, work-stations will be needed for on-line and off-line analyses and high-resolution graphics. Apollo is a candidate; LISA II is being evaluated by DELPHI; and there is the hope that DEC might show up with a competitive offer based on a microVAX running microVMS.

8. CONCLUSION

There is much activity at CERN and in the participating home laboratories. We have to find solutions for our communications problems which are caused by the distributed development. A lot of work still needs to be done, in particular in FASTBUS, VME, and for the emulators. The software problem has not been addressed at all in this presentation, which does not mean that it is solved! In fact, the opinions there are as divergent as the hardware solutions.

Fortunately, there are more than four years left, so there should be time to get the work done.

Acknowledgements

I would like to thank all the members of the four LEP collaborations as well as the people from the LEP machine, who provided the necessary input for this talk and helped with drawings and transparencies.

* * *

References

- [1] ALEPH Collaboration, Technical Report, CERN/LEPC/83-2, May 1983.
- [2] The OPAL detector, Technical Proposal, CERN/LEPC/83-4, May 1983.
- [3] Technical Proposal L3 Collaboration, CERN/LEPC/83- , May 1983.
- [4] DELPHI, Technical Proposal, CERN/LEPC/83-3, May 1983.
- [5] ALEPH Collaboration, Data acquisition and data analysis, CERN/LEPC/84-8 (1984).
- [6] DELPHI Collaboration, Data acquisition and analysis in the DELPHI Experiment, CERN/LEPC/84-3 (1984).
- [7] L3 Collaboration, Trigger and data acquisition system of L3, CERN/LEPC/84-5 (1984).
- [8] OPAL Collaboration, Report on data acquisition and data analysis, CERN/LEPC/84-4 and 84-2 (1984).
- [9] R. Hemingway, private communication.
- [10] H. Müller, The GPM, a general purpose programmable FASTBUS master (F6808), CERN, EP Division, 20 May 1984.
- [11] T. Lingjaerde, XOP, DELPHI 82/45 ELEC; A second generation fast processor for on-line use in high energy physics experiments, Proc. Topical Conf. on the Application of Microprocessors to High-Energy Physics Experiments, Geneva, 1981 (CERN 81-07, Geneva, 1981), p. 454.
- [12] A.K. Barlow et al., The UTI-NET book, CERN/SPS/ACC/Tech. Note 84-1 (1984).
- [13] H. Braffman et al., A fast general purpose IBM hardware emulator, Proc. of the Three Days In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padua, 1983, (University, Padua, 1983) p. 71.
- [14] P.F. Kunz et al., The 3081/E processor, same Proc. as Ref. [13], p. 84.
- [15] I. Gaines, The Fermilab Advanced Computer Program.
H. Areti et al., ACP Modular Processing System: Design specifications.
T. Nash, private communication.

Table 1
General LEP parameters

Machine circumference	26,658.879 m
Average machine radius	4.243 km
Minimum machine radius	4.204 km
Maximum machine radius	4.263 km
Bending radius	3.104 km
Number of intersections	8
Number of bunches per beam	4
Horizontal betatron wave number	90.35
Vertical betatron wave number	94.20
Momentum compaction factor	1.928×10^{-4}
Harmonic number	31,320
RF frequency	352.21 MHz

Table 2
Trigger summary and rates

Beam crossing: 22 μ s

TRIGGER	ALEPH	OPAL	L3	DELPHI	Unit
Level 1	1.5 ≤ 500	18+4 < 4	10 ~ 100	2 1000	μ s Hz
Level 2	50 ≤ 10	N/A	100 ~ 10	35 ≤ 20	μ s Hz
Level 3	N/A	10 ~ 0.4	5 10	15 5	ms Hz
Event processor	~ 2	?	a few	≤ 2	Hz
No. of channels	$\sim 300,000$	86,000	$\geq 50,000$	$\sim 150,000$	
Event size *)	$\sim 100,000$	140,000	70,000	200,000	bytes
Time per tape	15	30-60	30	4	min
No. of tapes	100	50-25	35-50	$\leq 400 !$	per day

* Expected running time: 100 days/year
* 20 tracks and 20 photons.



Fig. 1 Aerial view of the LEP site

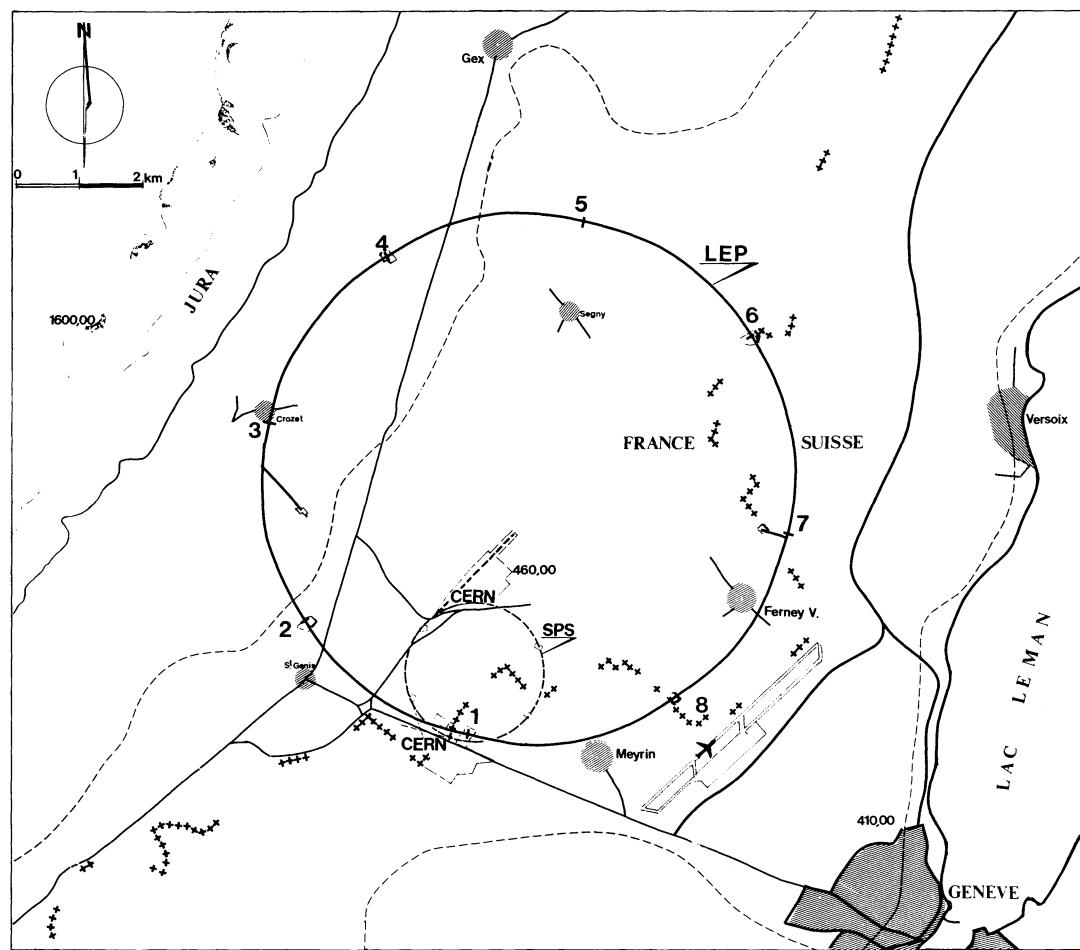


Fig. 2 Location of the LEP access points

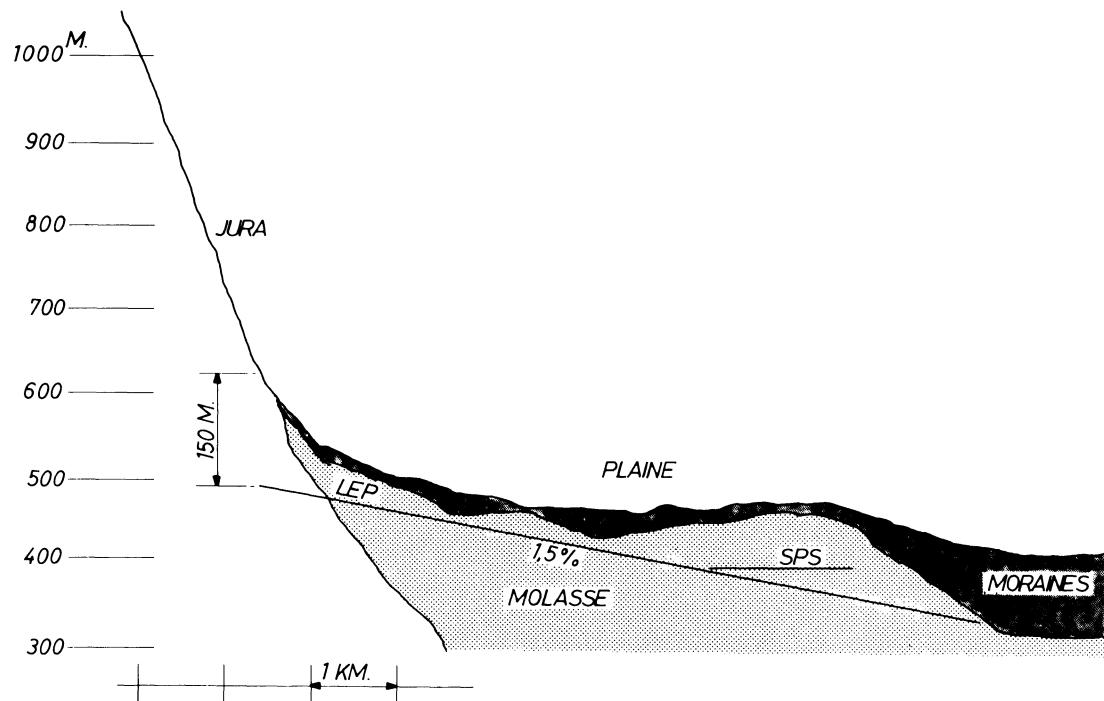


Fig. 3 Simplified geological section

H. Léonard

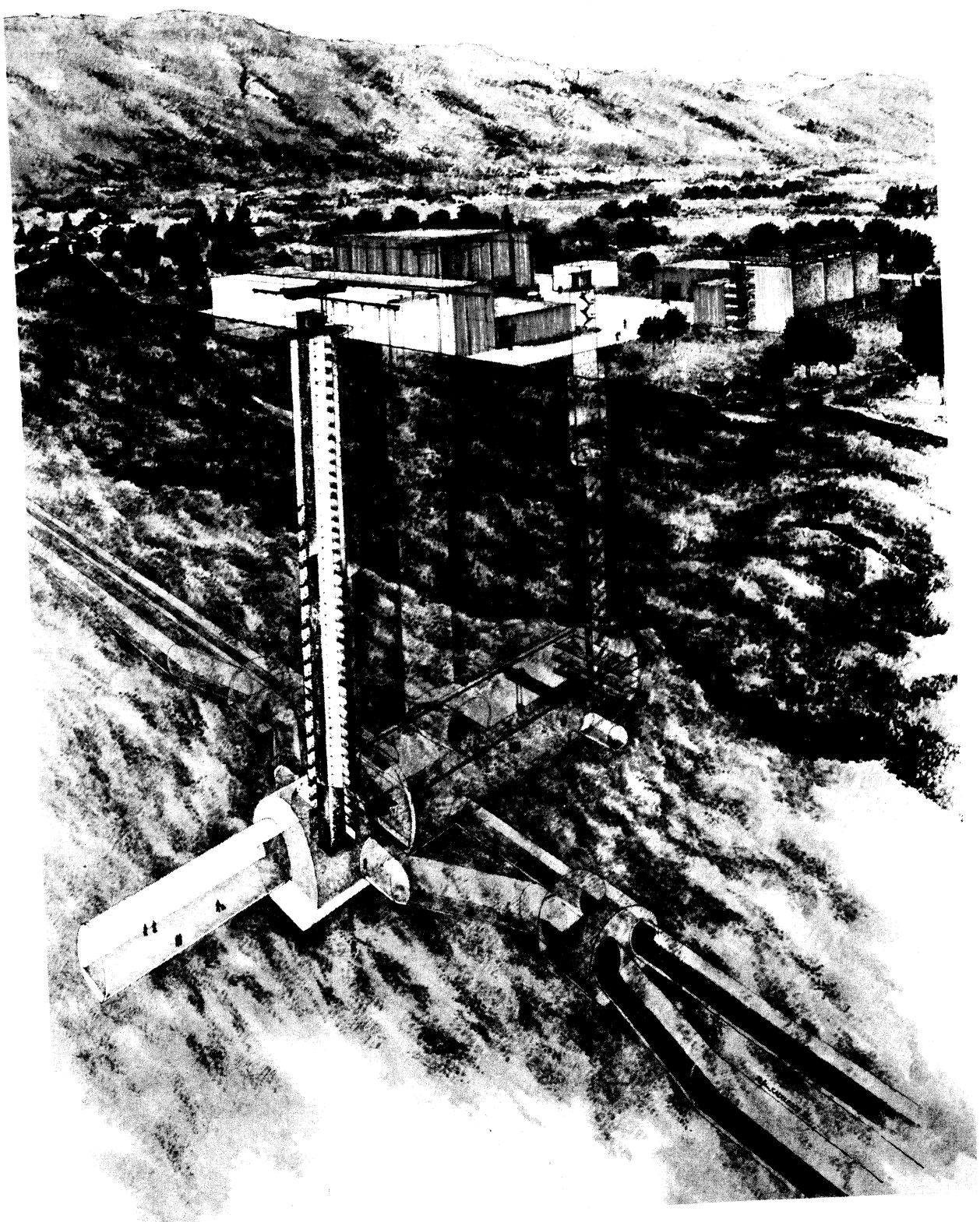


Fig. 4 Layout of an interaction region

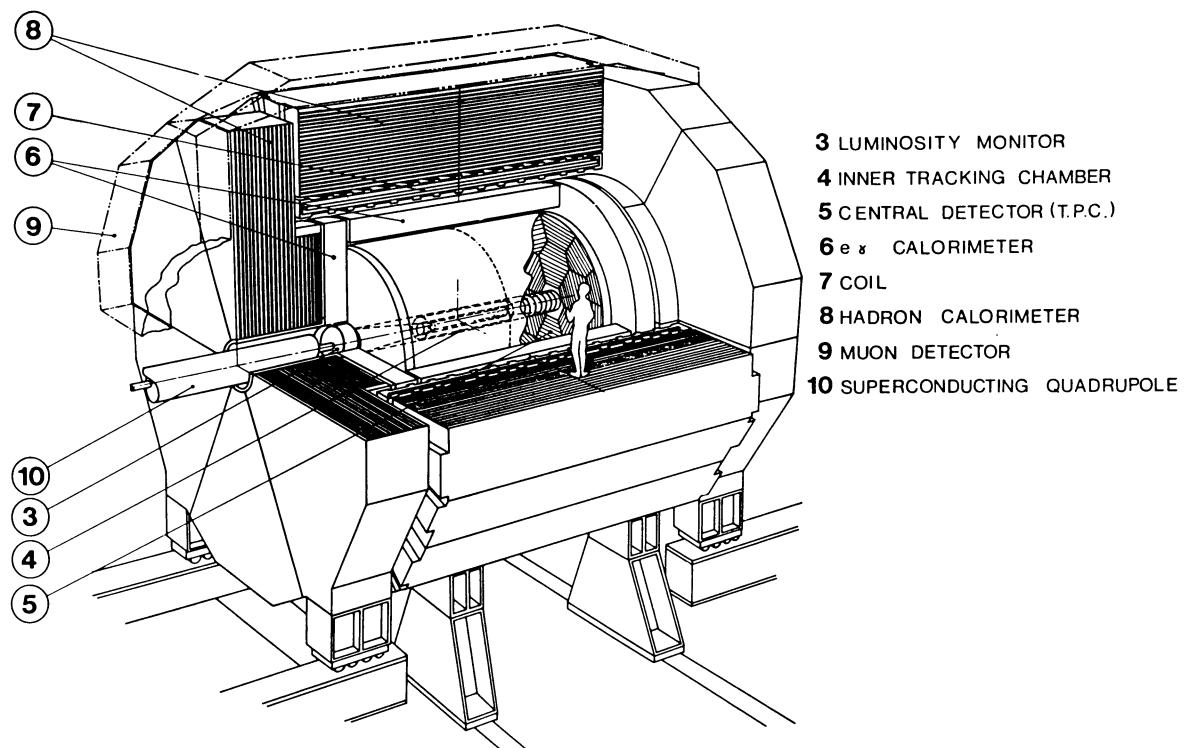


Fig. 5 The ALEPH detector

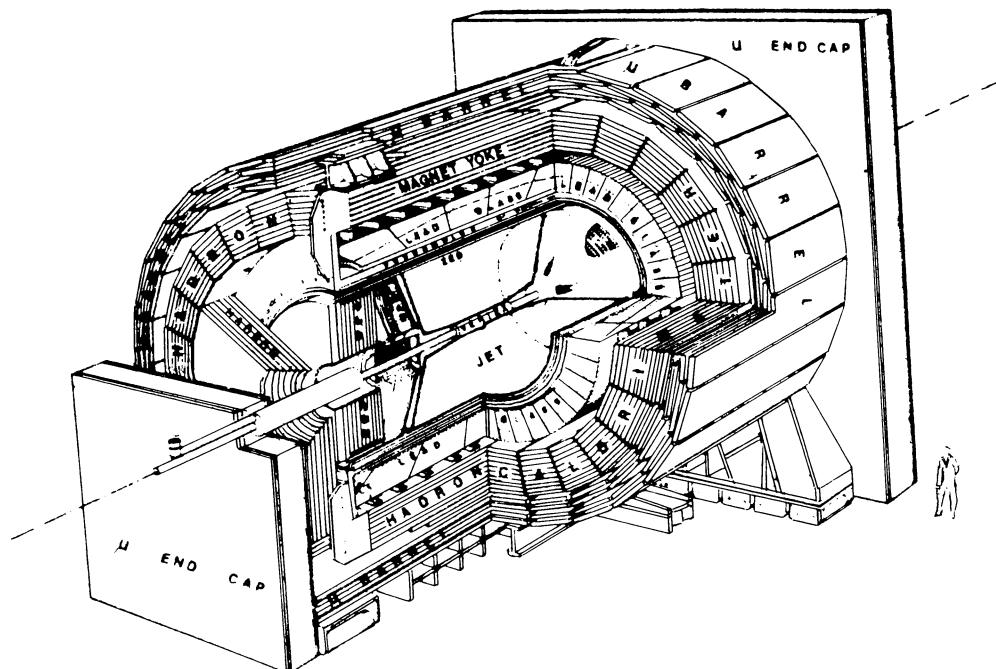


Fig. 6 The OPAL detector

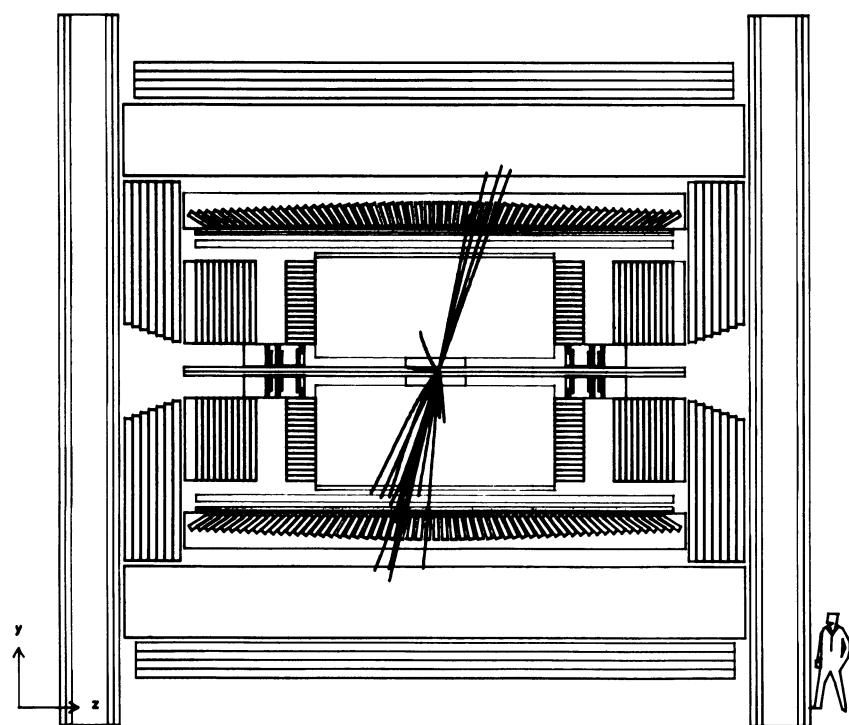


Fig. 7 A two-jet event in the OPAL detector (side view)

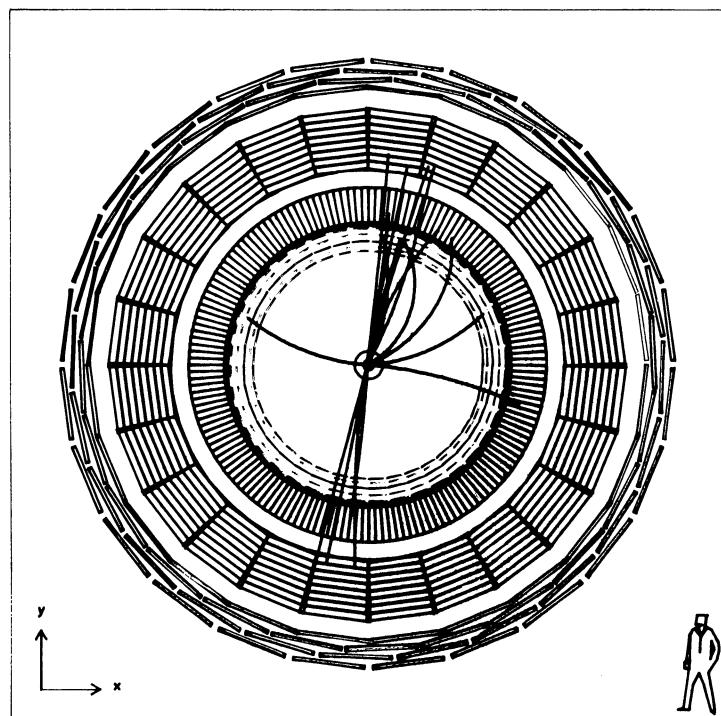


Fig. 8 Same event as in Fig. 7 (front view)

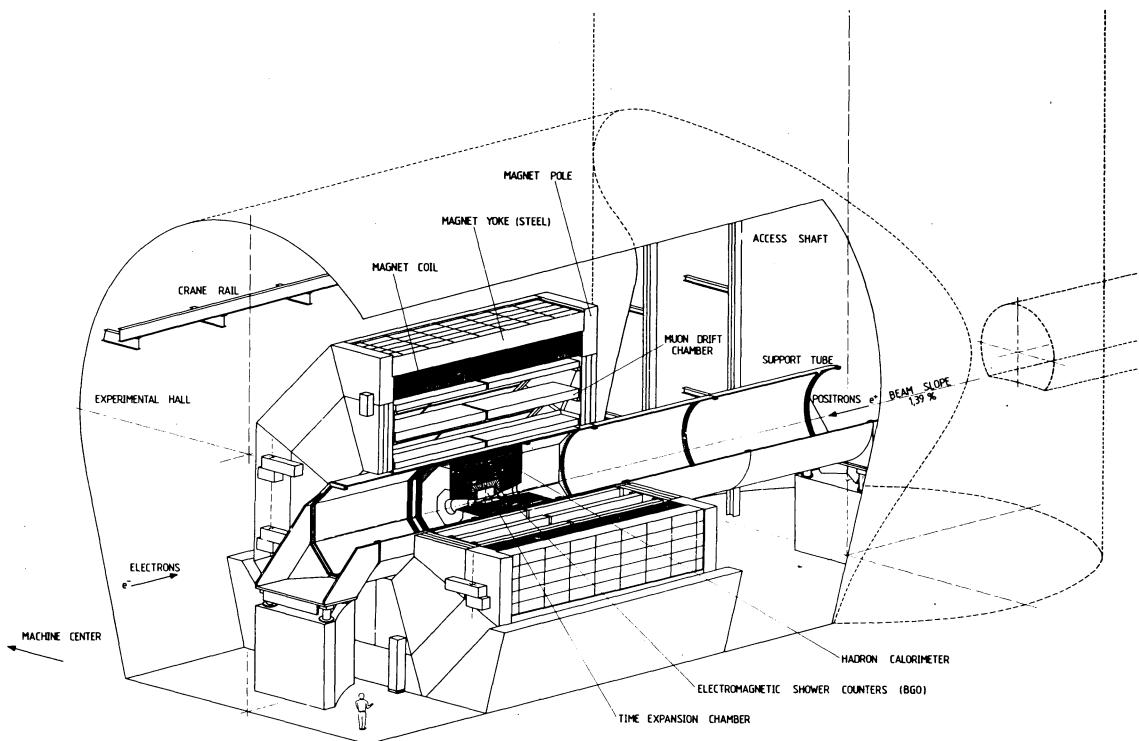


Fig. 9 The L3 detector

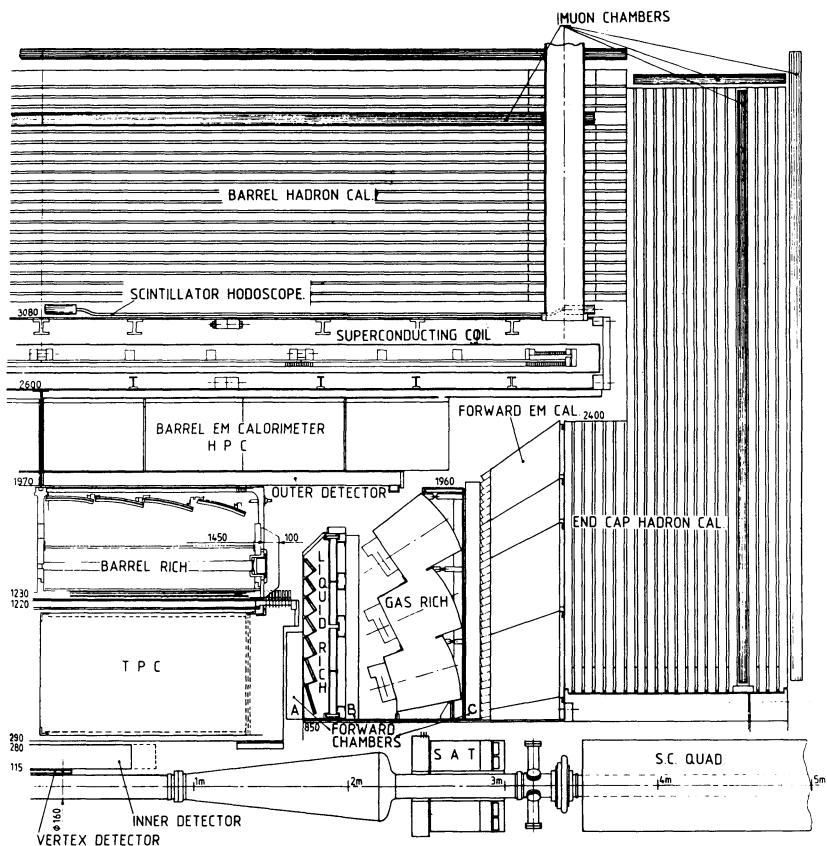


Fig. 10 Longitudinal view of the DELPHI detector

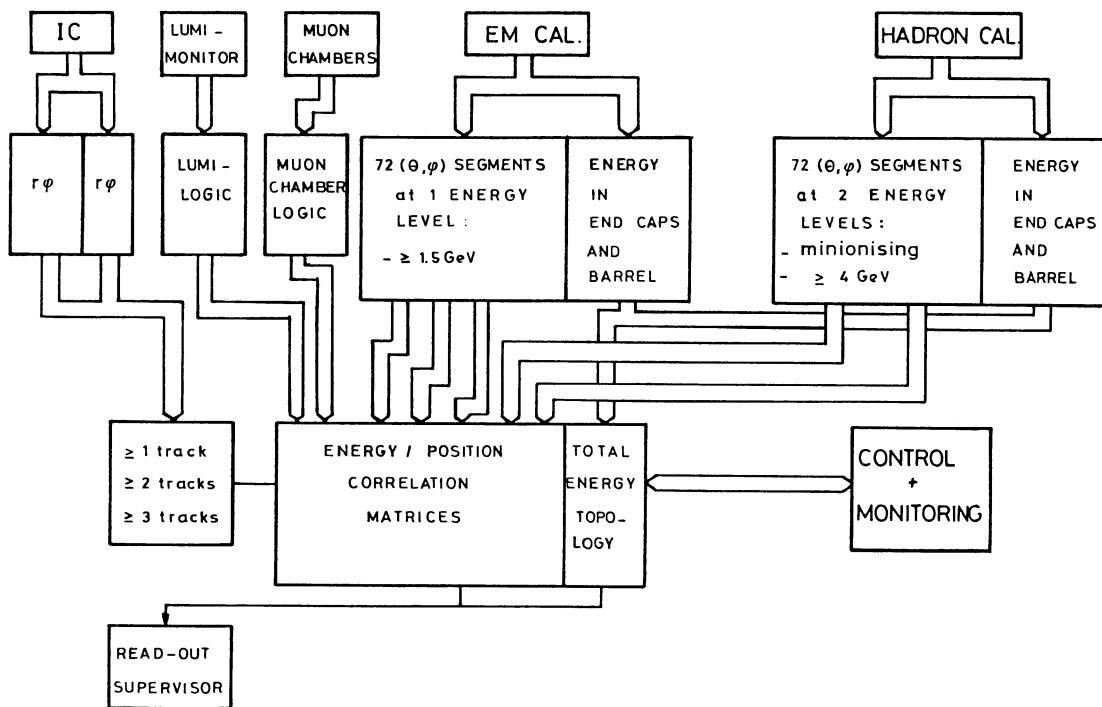


Fig. 11 Level 1 trigger logic for ALEPH

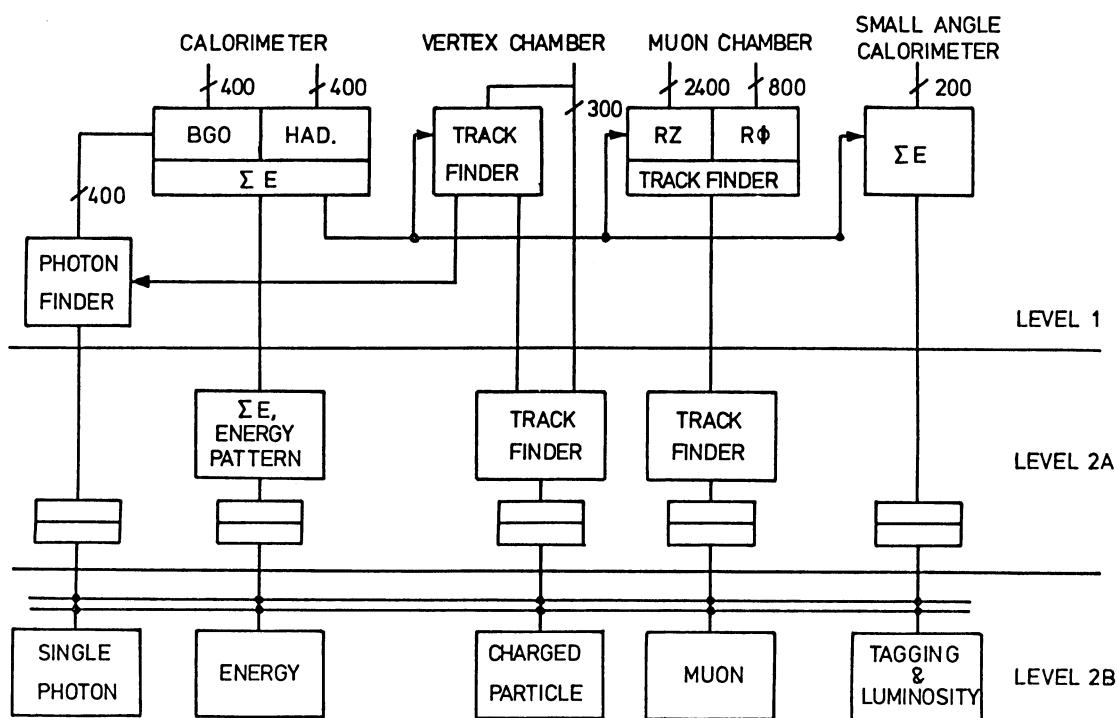


Fig. 12 Details of level 1 and 2 triggers for L3

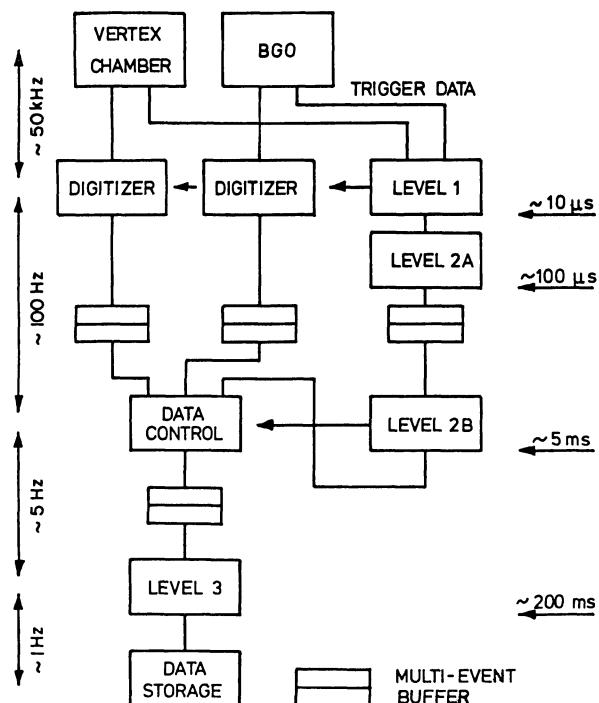


Fig. 13
L3 trigger and data acquisition system
block diagram

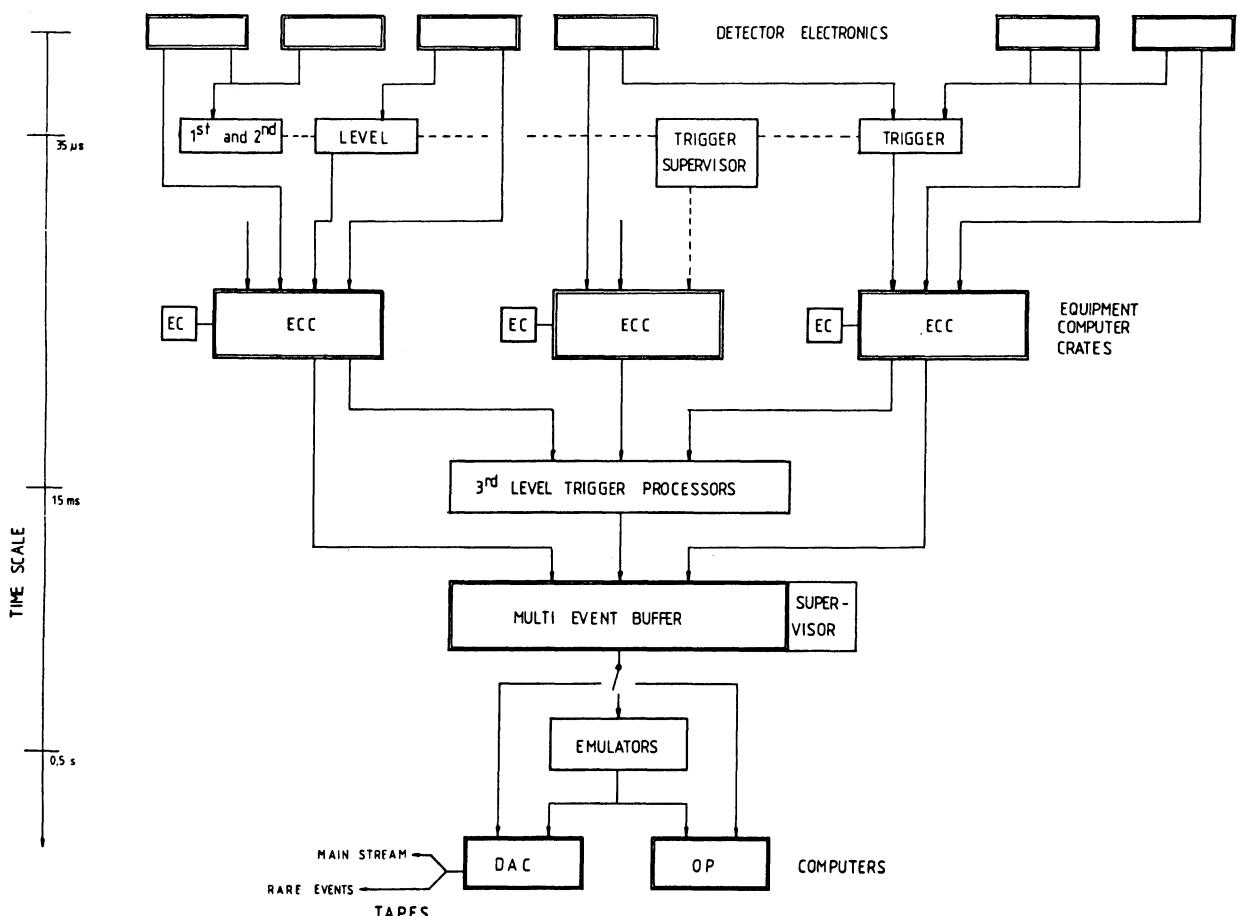


Fig. 14 Schematic diagram of the architecture of the data acquisition and filter system
for DELPHI

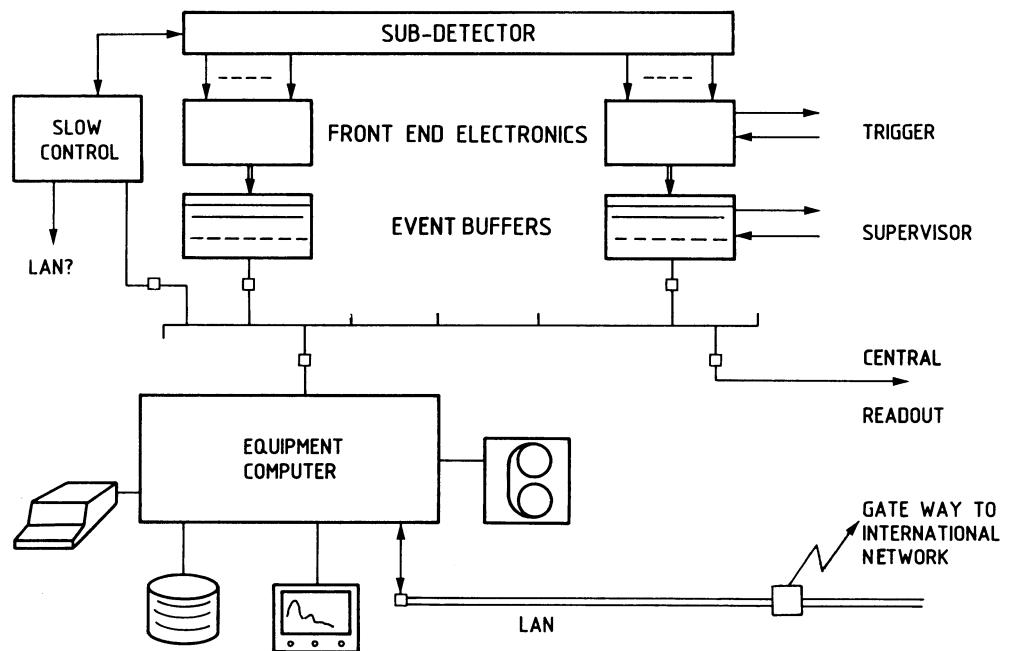


Fig. 15 A typical ALEPH sub-detector

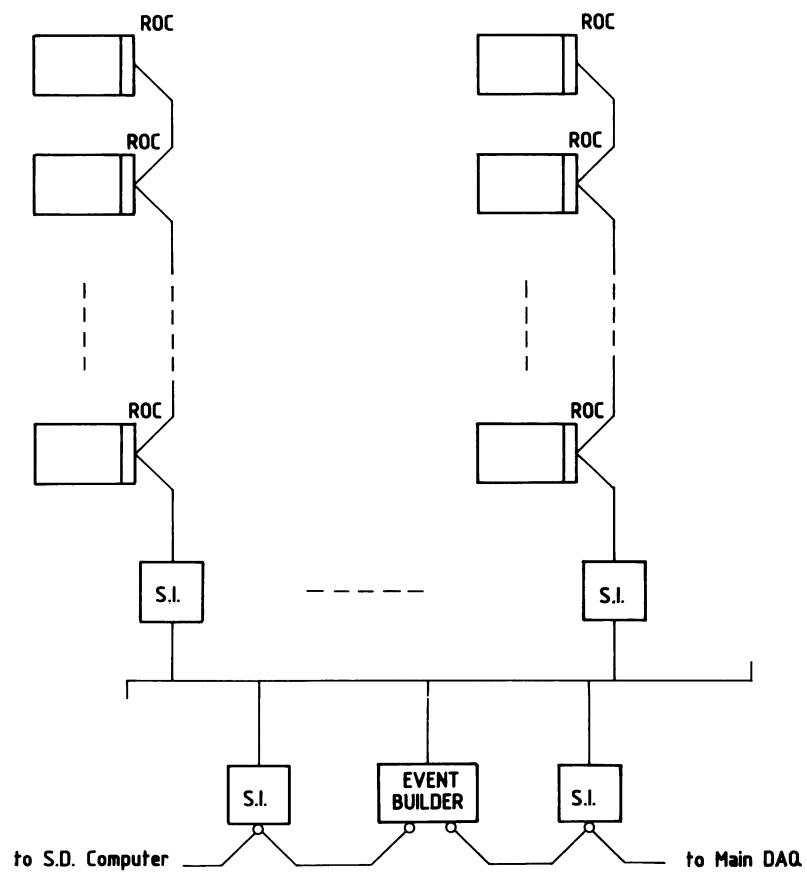


Fig. 16 ALEPH sub-detector read-out

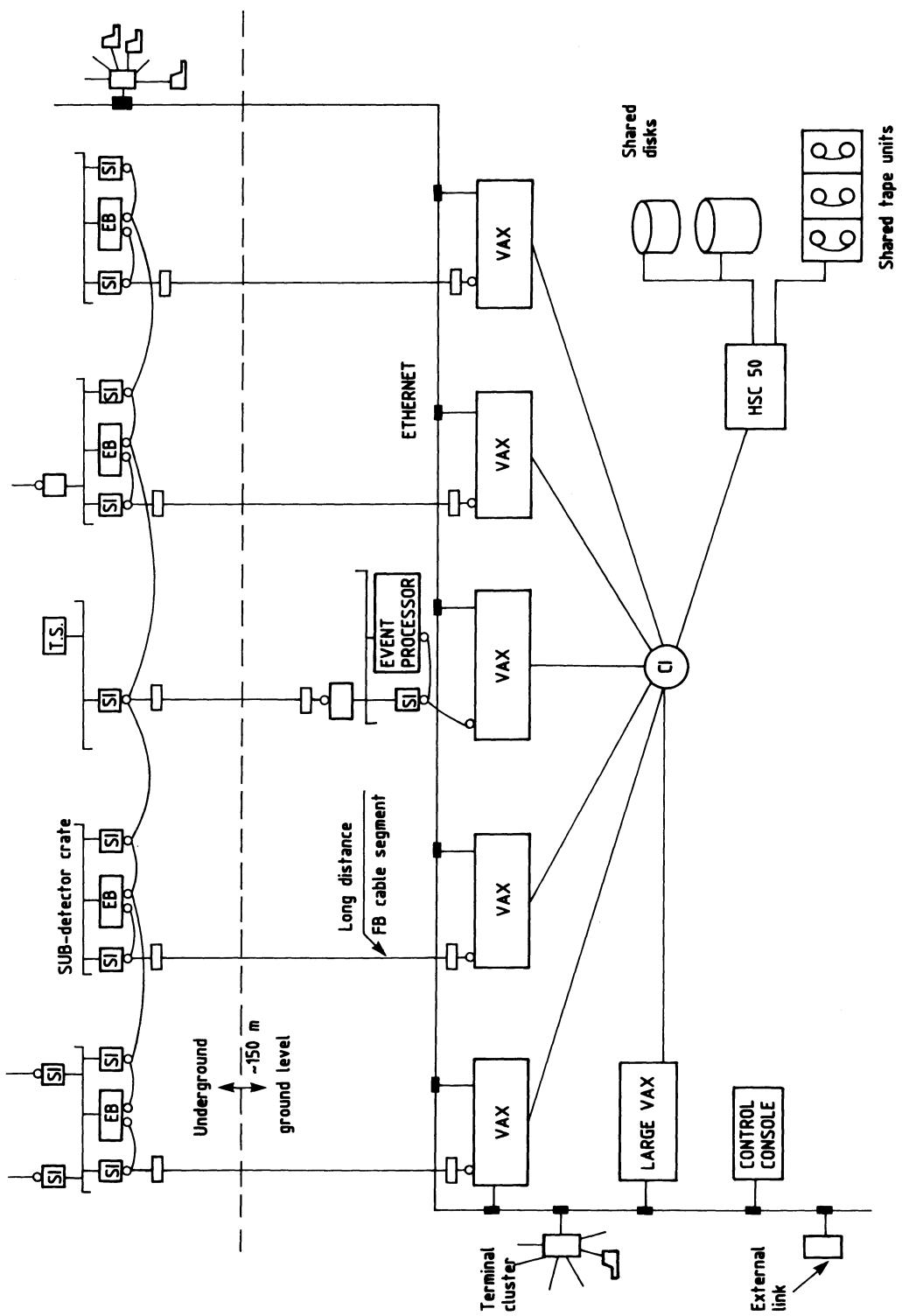


Fig. 17 ALEPH central read-out (preliminary)

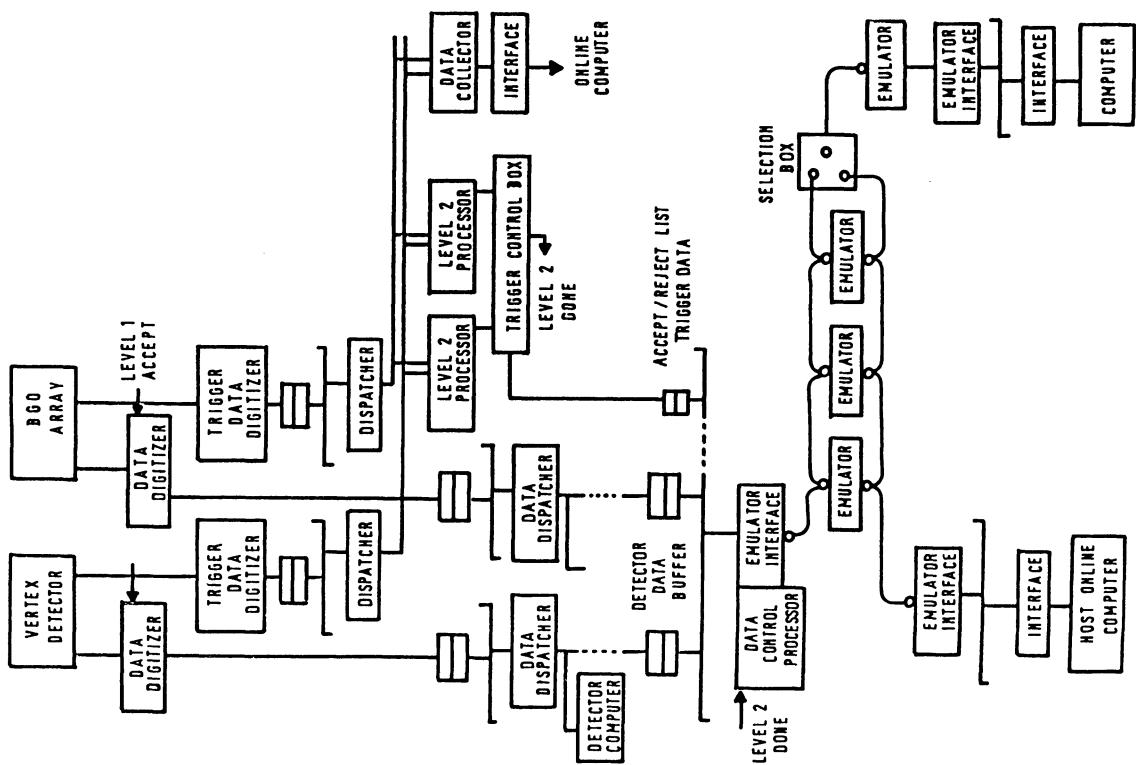


Fig. 19 L3 data flow

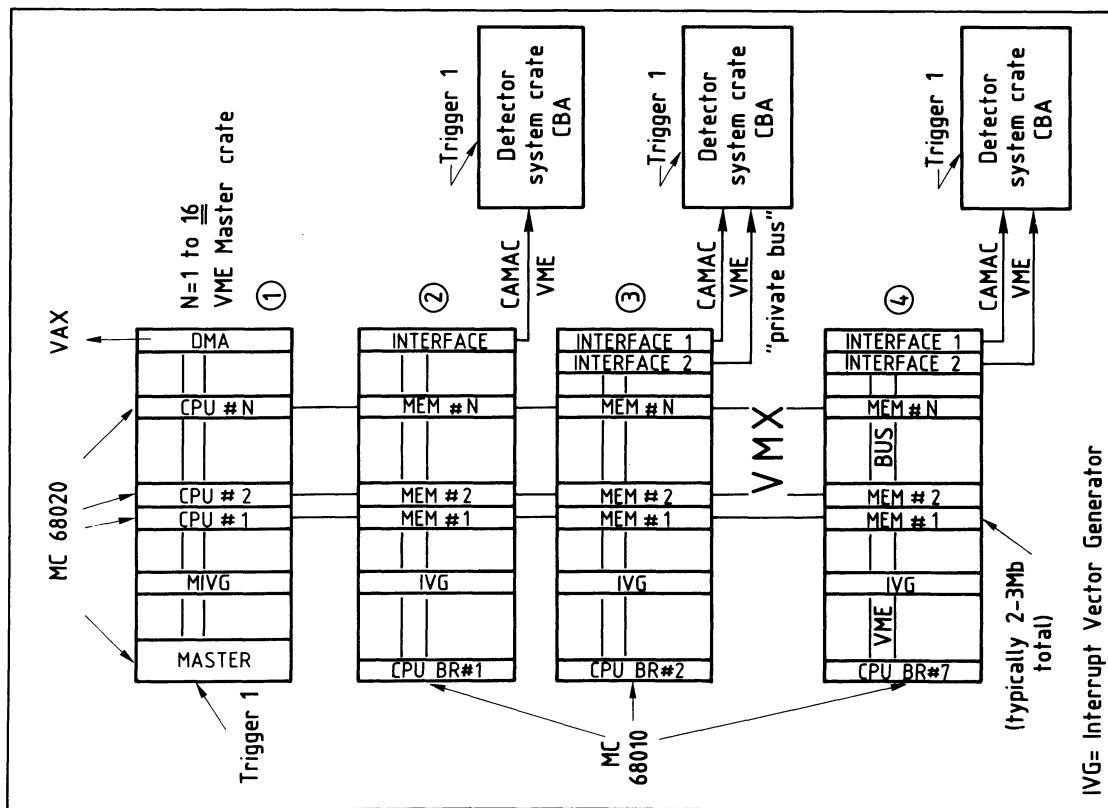


Fig. 18 OPAL VME interface and second level trigger

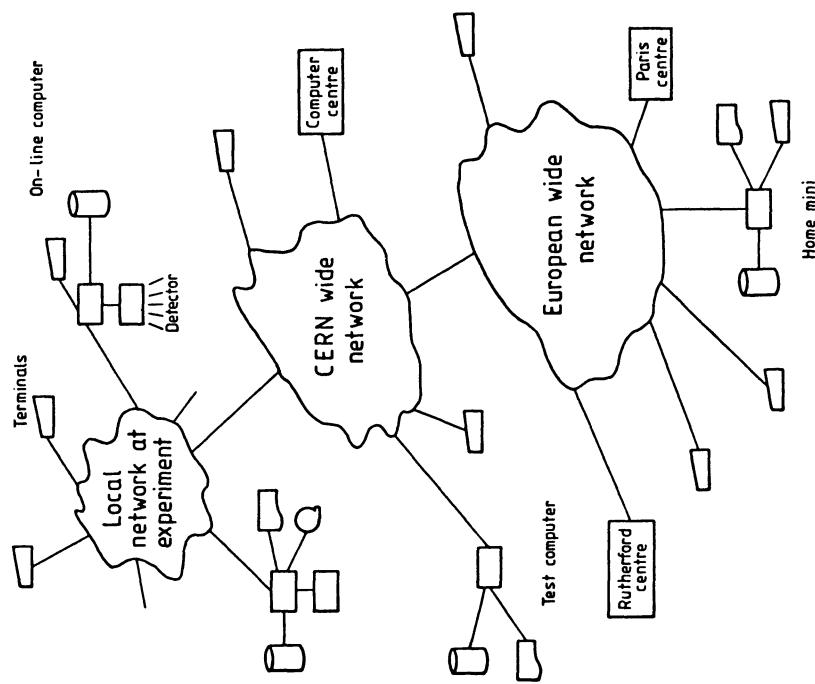
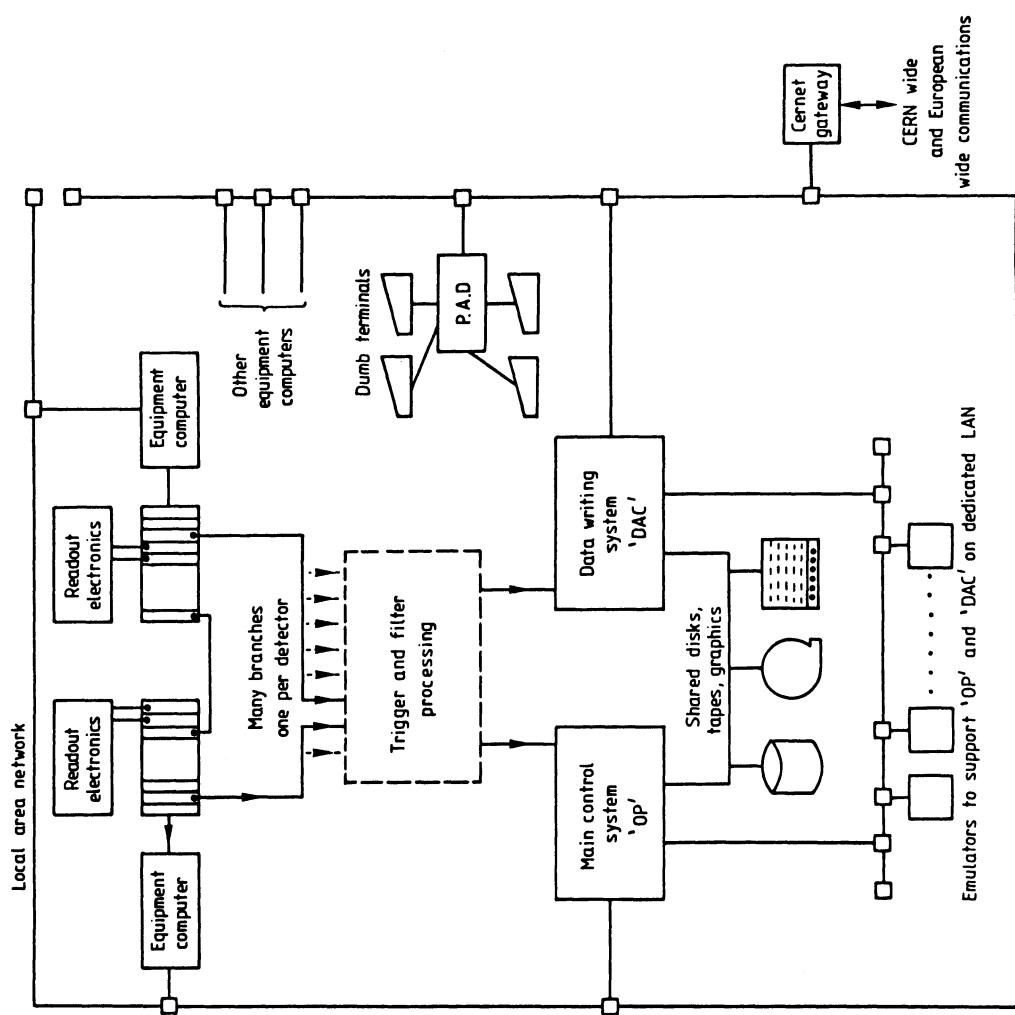


Fig. 21 Levels of networking required by DELPHI



DELPHI on-line system

Fig. 20 Overall system architecture for DELPHI computer control/data acquisition system

DATA NETWORKS AND OPEN SYSTEMS

R A Rosner

University of London Computer Centre, UK

ABSTRACT

Computing in the LEP era will require a variety of communications facilities, ranging from high-speed local area networks forming the backbones of distributed control systems to wide area networks connecting data analysis centres together. The ISO model for Open Systems Interconnection (OSI) offers a possible framework for the general study of communications environments, whatever their performance parameters or geographical extent. This series of lectures uses the model as the basis for discussing elements of the communications hierarchy likely to be required for LEP computing. Examples are given of the practical application of OSI principles to real communications problems.

1. INTRODUCTION

From its earliest days, high energy physics has persisted in making great demands on all aspects of the available technology, often pushing it to its very limits. The construction of LEP is taking place at a time of feverish activity in the field of data communications and, of course, LEP experiments will require the best of what is available, if not better. A catalogue of these communications requirements includes very high bandwidths over short distances all the way up to more modest rates around the globe. LEP communications therefore provides us with a useful context in which to study a wide variety of techniques which can be combined together to serve a single objective.

Building such a communications system from various somewhat disparate technologies carries with it the danger that the individual pieces of the jigsaw might not fit together very well. It is therefore important to step back and obtain an integrated view of the total communications system. As with any respectable branch of science, an abstract model can assist our understanding. The use of an abstract model for communications systems could serve as a uniform framework into which, if the model is any good, each of the various components could be slotted. One possible approach is the Open Systems Interconnection (OSI) model proposed by the International Organisation for Standardisation (ISO). These lectures are intended to explain the main features of the OSI model and to show how different technologies relate to it. Hopefully, the reader will derive an understanding of communications systems and their jargon sufficient for application to real problems.

No attempt is made at complete coverage of this rapidly expanding field. Emphasis is rather placed on the selection of typical illustrative examples. Some pointers are given

in the bibliography to more complete and detailed coverage of particular topics. The reader should be warned that the treatment here is unashamedly biased towards an integrated systems approach. Those favouring either greater objectivity or descriptions of ad hoc solutions are advised to look elsewhere.

2. COMMUNICATIONS AND LEP EXPERIMENTS

Von Ruden's lectures in this volume cover the online computing aspects of LEP experiments. The salient features of a typical experiment are listed in table 1 and the main phases are shown in figure 1. The communications requirements are given in table 2. The apparatus will consist of a very large number of components and data rates will be high. The complexity of the processes under study will necessitate sophisticated triggering mechanisms in which extensive computations will be performed on primary data even before the full information on candidate events is read out. The limited time period during which the apparatus is sensitive to the presence of charged particles demands that large volumes of data are collected centrally for decision-making logic to work on. As described by Von Ruden, this process will involve the use of high speed buses working in the 100 Mbit/s region within a few metres of the associated experimental equipment. Once the main data has been read out, local filtering will remove unwanted noise and the remainder will be transmitted at around 10 Mbit/s from the various parts of the equipment to the control room. Because of radio-activity, this could be about 100 m away. Local area networking techniques such as Ethernet would be appropriate. In the control room itself, each major item of equipment will have its own dedicated online computer, responsible for processing the incoming data. A single master computer will receive the results, record them on magnetic tape (or perhaps a denser medium) and perform housekeeping functions. Again some form of local area network will link together the computers in the control room.

The physicist's confidence in the quality of his recorded data can only be maintained by performing very detailed computations on a small sample of events. These calculations are akin to the full analysis through which all the recorded events will eventually have to pass. The sampling will usually be done as a background task by one of the online computers in the control room. In some cases, it may be necessary to make use of the considerably greater computing power available at the CERN computer centre. This could be up to 10 kms away and a "metropolitan" network offering transmission speeds in the region of 100 kbit/s is envisaged as the connection mechanism. This network will also

Table 1

A typical LEP experiment

Cost SFR 70M
500 man-years effort
100,000 electronic channels
14 major items of apparatus
~5 online computers (e.g. VAX)
~200 microprocessors
Data collection period ~2000 hrs per year
Candidate event rate ~25 per minute
Average event size (recorded data) ~100 kbytes
Data per hour ~150 Mbytes (1x6250 bpi tape)
Data per year ~2000 tapes
36 institutions

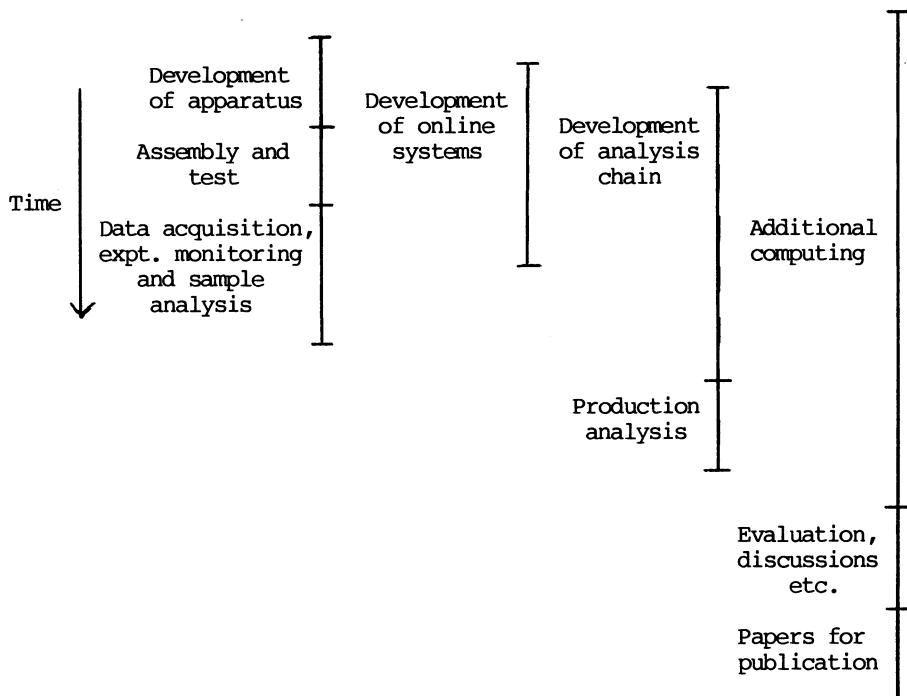


Fig. 1 Phases of an experiment

Table 2
Communications requirements of an experiment

Function	Bandwidth	Distance
Raw data from individual experimental components	100 Mbit/s	10 m
Filtered data to online computers	10 Mbit/s	100 m
Processed data for central recording	1 Mbit/s	10 m
Sample analysis using on-site mainframes	100 kbit/s	10 kms
Apparatus monitoring and control	10 kbit/s	100 m
Exchange of data with LEP control system	10 kbit/s	10 kms
Sharing of facilities among on-line computers	1 Mbit/s	10 m
Development of analysis chain	10 kbit/s	1000 kms
Electronic messaging and teleconferencing	10 kbit/s	1000 kms
Remote monitoring and control	10 kbit/s	1000 kms

serve to distribute relevant parameters about LEP itself among the control rooms of the various experiments.

LEP experiments will always be collaborative ventures among physicists from several institutes. In the DELPHI experiment, for example, the participants come from one US and thirty-five European institutes distributed all over the continent. National and international wide-area communications will be needed so that participants can work together, obtain up-to-date information on the status of their experiments and cooperate on the analysis software. Speeds of around 10 kbit/s across the continent will be adequate though it might be useful occasionally to transfer a complete tape-load of events at somewhat higher speeds.

The high cost of machines like LEP and of the experiments performed on them is having a profound effect on the sociology of high energy physics. University experimenters are forced to divide their time between duties at their home bases and periods at the accelerator laboratory. Only efficient communications mechanisms will prevent them from losing touch with developments on their experiments while they are at their universities. It has even been suggested in some quarters that facilities should be available for the remote control of experiments by absentee physicists but this may not yet be a practical idea!

3. CONNECTIVITY AND INTERCONNECTION

The previous section indicated a range of communications features. LEP is not special in this respect; it merely demonstrates in a single example characteristics which it has in common with a large number of other applications. These are:-

i) Flexibility of connections

There should be no technical obstacles to the establishment of communications paths among users, terminals and computer systems.

ii) Range of distances and rates

Requirements will vary from rates of 100 Mbit/s over a few metres to 10 kbit/s over distances of the order of 1000 kms.

iii) Variety of equipment

The equipment involved in communications will include terminals, micros, personal computers, conventional minicomputers and mainframes. These will be purchased from a large number of different manufacturers.

iv) Dispersed development

In addition to the computing equipment purchased off-the-shelf, ventures such as LEP will also be concerned with the development of special-purpose apparatus. Inevitably, the responsibilities for such developments will be split among several geographically dispersed institutions.

v) Interworking

Meaningful communications among heterogeneous devices from different suppliers can only be achieved by the use of agreed conventions for interworking. In precisely the same sense, a complex piece of apparatus consisting of components developed in different places will only function if they intercommunicate via pre-defined rules.

At this point, a distinction is drawn between connectivity and interconnection.

Connectivity is defined as the capability for bits to be transmitted between any pair of entities. Interconnection is the property which both parties to a transmission must share if a common interpretation is to be given to the bit sequences exchanged.

The following section explores some of the mechanisms available for achieving connectivity.

4. DATA NETWORK CONCEPTS

It would obviously be possible to satisfy all our communications needs by laying direct links to all the destinations with which we should ever want to exchange information. Such a solution would clearly be as ludicrously impractical as it would be expensive. Dr Johnson, an English literary figure of the 18th century, defined a network as

"anything reticulated or decussated at equal intervals with interstices between the intersections."

Abandoning such pomposity, we take as our example the telephone network which achieves economies by providing for any pair of subscribers to call each other regardless of whether they should ever need to do so. Data networks achieve the analogous functions of transmission and switching for computer-oriented communications.

4.1 Packet Switching

In the mid-1960's, it was suggested that the emergence of relatively cheap minicomputers could be used to achieve advances in long-distance data communications. Point-to-point links were expensive and, by its very nature, computer traffic was not suited to the permanent utilisation of dedicated bandwidths. The computer's natural mode

of handling data is in blocks of up to say 4K bytes. Furthermore, computers by virtue of their multiprocessing and timesharing capabilities can sustain several independent communication streams simultaneously. The idea of packet-switching was to replace a mesh of point-to-point links by a limited number of "trunk" links and exchanges based on small computers to form a network, as shown in figure 2. Each subscriber system (A, B or C) is connected to the network by a single link. Traffic between a pair of subscribers (e.g. host computers A and B) is fragmented into a sequence of packets AB1, AB2, AB3 etc. each of maximum length say 128 bytes defined by the network. The packet-switching exchanges of the network are given sufficient information in a packet sequence to determine an appropriate route for the data.

The main attributes offered by packet-switching networks are:-

- i) The bandwidth of any link can be allocated dynamically without prior reservation for any particular call. In this way, the effective utilisation of expensive long-distance links can be maximised.
- ii) Traffic to and from each subscriber is carried by a single cable thereby reducing the requirement for a large number of expensive ports on host computers. (Of course, large traffic volumes and limited bandwidths may lead to the need for some subscriber hosts to have more than one link to the network.)
- iii) There may be mismatches of processing speeds between a pair of communicating subscribers either because of inherent technical differences or because of high traffic flows to a particular destination. The storage available in the memories of the packet-switching exchanges can be used to buffer packets in transit. Thus the flow of traffic to a destination which is unable to keep pace with the rate of input into the network can be controlled and the source throttled back.

4.2 Packet Assembler/Disassemblers (PAD's)

So far, we have discussed traffic between computer systems capable of handling information in "packets". Most simple terminals can only send or receive single characters and cannot therefore be directly attached to packet networks (quite apart from the expense of so doing!). Devices known as PAD's (Packet Assembler/Disassemblers) have been invented to convert between the character streams of such terminals and the packet streams of the network. The concept is illustrated in figure 2 which also indicates that a number of terminals can be connected to the network via a single PAD thereby reducing costs considerably.

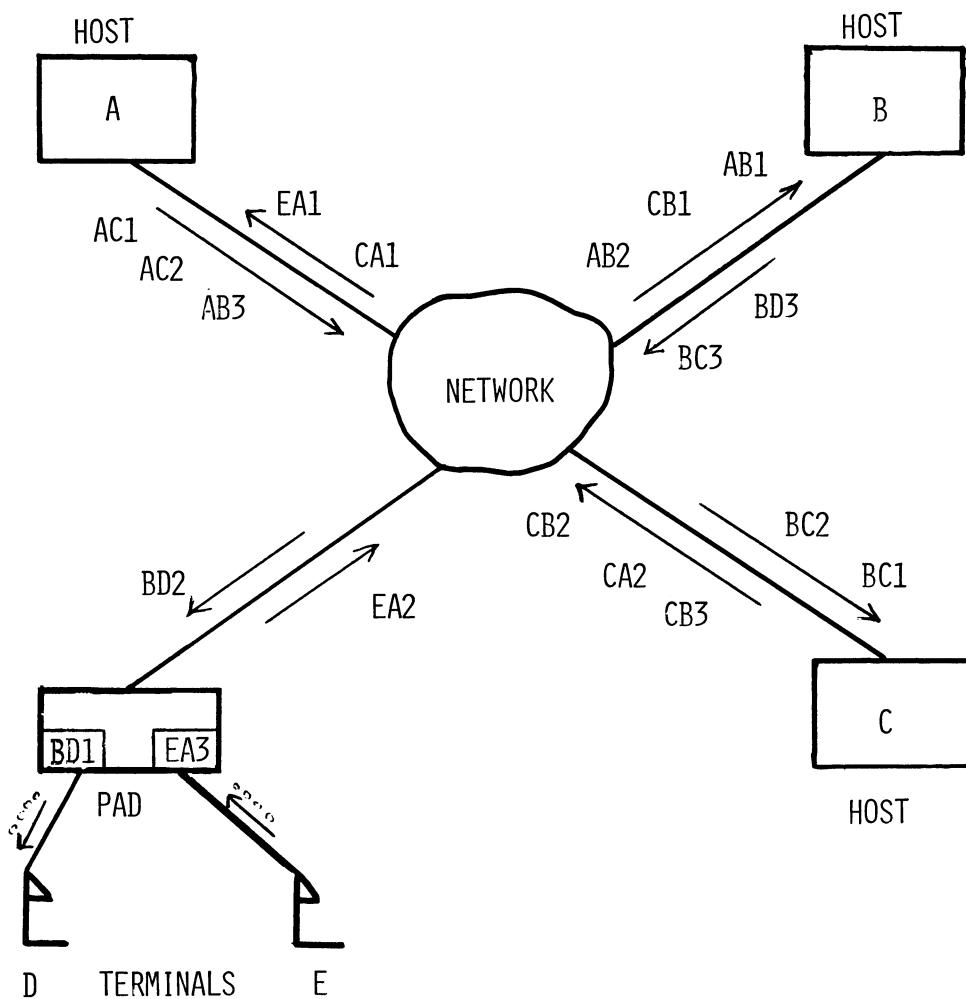


Fig. 2 Packet switching principles

4.3 Datagrams and Virtual Circuits

The implementation of a packet-switching data network can follow one of two basically different philosophies known as datagrams and virtual circuits. In datagram networks, every packet is independent and carries complete information about its source and destination to enable the exchanges to carry out the routing function. No facilities are available in the network itself to ensure that packets are delivered in the order of transmission and that there is no loss or duplication. Nor can the source be informed by the network that the transmission of a packet has been successful and error-free. In other words, packets are injected into the network and the transmitting systems hope for the best. If they require more sophisticated facilities, these have to be constructed within the subscriber systems themselves over the basic datagram mechanism.

In contrast, virtual circuit networks require a "virtual call" to be established between communicating parties by means of a special packet transmitted by the calling subscriber at the outset. This contains source and destination addresses and various other parameters needed for the call. From then on, a unique code is assigned to the call so that all the packets belonging to it can be identified without each having to repeat the source and destination addresses. All packets have sequence numbers and the network takes responsibility for delivering them in the correct order. If this is not possible, the source is informed of losses or transmission errors so that synchronisation can be re-established. The sequence numbers are also used to control the flow so that packets are only delivered if the receiver is capable of handling them.

The letter post system is a datagram network while the telephone network operates by virtual circuits.

In the 1970's, a furious debate of theological proportions raged between the datagram and virtual circuit factions. This is not the place to repeat the arguments. The powerful PTT's were firmly in the virtual circuit camp and it is no surprise that public data networks are of this type. The use of virtual circuits allows the network administration to control the allocation of total network resources more easily and to apportion the charges for network usage more readily. X25 is the by now famous technical standard for public virtual circuit networks and has also been adopted for many private networks. However, ARPANET and CERNET are datagram networks. As with most dogmatic wars, it has become evident with the passage of time for cooler reflection that neither side can claim complete victory in all cases. There are circumstances more suited to one or

other technique. The issue has been raised again with the advent of high speed local area networks. This time, the old wine has been put into new bottles under the label of connectionless (datagram) vs connection-oriented (virtual circuit) transmission.

5. WIDE AREA NETWORKS

5.1 Network Access

X25, the network access standard, is so important and widespread that it deserves a subsection to itself. There has been much misunderstanding about X25 resulting from the fact that it embodies not just one but three standards. These cover three different aspects of connecting a subscriber to a packet network and this structure will be dealt with later in these lectures. For the moment, our attention is focussed on X25 level 3 which covers network access procedures. As usual with international standards, it consists of syntax defining the packet formats and encodings followed by semantics which describe the dynamic behaviour of the protocol. Formally, this can be viewed as a set of state transitions; in other words a set of instructions to implementors on the actions to be taken under various circumstances.

Each subscriber on a network is referred to as a DTE (Data Terminal Equipment) and has a unique address assigned to it by the network administration. Its physical link to the network is conceptually divided into 4096 logical channels each capable of sustaining an independent conversation with some other DTE. Once a call is established, it is assigned a logical channel number (LCN) which is of purely local significance between a DTE and its nearest network exchange. For outgoing calls, the DTE chooses the number of a free channel while for incoming calls the network exchange makes the choice. The rules governing this selection mechanism are clearly defined by the network administration. It must be emphasised that, for a given call, there is no relationship between the logical channel numbers at either end of the call. How the network joins the two together is no concern of ours.

The sequence of events in an X25 call is roughly as follows. The calling DTE first issues a CALL REQUEST packet containing both its own address and that of the called DTE. Also included is the calling DTE's choice of logical channel number. The packet-switching exchanges pass this packet through the network, reserving whatever buffers and other resources are necessary to sustain the call. Before the final hop, the last exchange chooses an LCN on its link to the called DTE and embeds this in the appropriate field of the CALL REQUEST packet. From now on, each DTE refers to the call only by the appropriate

LCN. The called DTE accepts the call and data packets can then be exchanged, each with its own sequence number.

One of the beauties of X25 (and of modern networking protocols in general) is that data packets in one direction can contain acknowledgments for data received in the stream flowing in the opposite direction. This is known as piggy-backing and is achieved by including the sequence number of the next expected packet.

Sequence numbers are allocated to packets cyclically, usually modulo 7. If a DTE has no data to send, it can still acknowledge receipt by sending a RECEIVE READY (RR) control packet with the sequence number of the next data packet expected. Alternatively, it can stem the flow by issuing a RECEIVE NOT READY (RNR) control packet and reopen the channel later on with an RR. If a packet is received out of sequence or contains errors, a DTE can send a REJECT (REJ) control packet to request retransmission from a specified sequence number.

The concept of a window allows several packets to be in flight simultaneously without the need for each of them to be individually acknowledged after receipt. The window size is the number of packets on a given call which can be sent into the network before an acknowledgment is required. This is a facility which can be negotiated between calling and called DTE's via the initial exchange of packets to establish the call. In practice, a good rate of data flow can be achieved by choosing a window size equal to the number of hops between the DTE's. This allows each of the links on the route to be simultaneously transmitting one packet of the call. (Of course, such a simple view breaks down if network congestion or failed components require alternative routing strategies to be invoked.)

If difficulties encountered during a call become so serious that resynchronisation of the DTE's is impossible, either DTE can request a RESET of the logical channel. After this, each DTE sets the window and the sequence number of its next packet in the call to zero so that both parties are once again in step. Failures of the network itself can also be signalled to the DTE's by RESET packets on the appropriate logical channel. Malfunctioning of a DTE or its network exchange may require the more drastic action of a RESTART, which is equivalent to a reset of all its active logical channels.

To end a call, one of the DTE's sends a CLEAR REQUEST to which the other DTE responds with a CLEAR CONFIRMATION. At this point, the resources of the network allocated to the call can be released and the charges added to the bill!

5.2 Dumb Terminals, PADs and Hosts

There are three standards known as X3, X28 and X29 (often compressed to Triple X or even XXX) used for the operation of terminals connected to X25 networks via PAD's. X3 is a list of parameters associated with each terminal on a PAD. Some of the parameters are relevant to interactions with a host, others are concerned with properties of the terminal itself. Each parameter has a value which can be set by the user at the terminal or by the host with which he is communicating. However, there are some parameters, such as those describing characteristics of the terminal, which must remain fixed.

X28 describes the interface between the terminal and the PAD, such as how the user can examine parameter values or change some of them. X29 describes the protocol of exchanges between a host and a PAD. This includes the means of examining and changing parameters and indicating by means of an identifier in the appropriate field of an X25 CALL REQUEST packet that an X29 call is being initiated from a terminal on a PAD.

5.3 Public Network Tariffs

Every X25 public data network has a scale of charges levied on its subscribers. These consist of two elements. Fixed charges cover initial installation of a link from the subscribers premises to his nearest exchange plus a rental for each three-monthly period. Usage charges depend on the volume of data sent and the duration of the calls.

6. LOCAL AREA NETWORKS

Strangely enough, it was for long-distance or wide-area data communications that the early advances in networking were made. The development of local communications techniques is more recent.

Local Area Networks (IAN's) are required to interconnect a large number and variety of devices. Traffic volumes may therefore be high. The cost of a port must be commensurate with the power and cost of the device to be attached to the network. There must be a scheme to ensure that each subscriber device is given fair access to the network.

In contrast with wide-area networks such as those discussed in the previous sections, local networks cover distances of up to about 1km and are usually run by a single organisation for its own use. The technologies available are capable of total bandwidths in the region of 10 Mbit/s though recent advances are pushing this closer to 100 Mbit/s. Even at such high speeds, error rates are several orders of magnitude lower than for wide-area networks.

We choose the Ethernet and the Cambridge Ring as two contrasting techniques to illustrate some of the concepts.

6.1 The Ethernet

A typical topology is shown in figure 3. The network consists of several interconnected segments of coaxial cable. The devices to be attached are denoted as stations in the diagram. They are connected to the coaxial cable by passive taps. The coaxial segments are linked via repeaters so that a signal on any segment is transmitted throughout the network. Each segment is suitably terminated to reduce reflections.

The access method is known as Carrier Sense, Multiple Access with Collision Detection (CSMA/CD) and enables control of the shared communications medium to be truly distributed among all the stations. There is no pre-allocation of time slots nor sharing of frequency bands. A station wishing to transmit is said to "contend" for use of the shared channel until it acquires access. Only then does it transmit its packet (or "frame" in Ethernet parlance). Access is equally fair for all stations - there is no prioritisation.

Before attempting to transmit, a station first checks whether the network is busy. If activity is detected (i.e. Carrier is Sensed), transmission is deferred. As soon as the channel becomes silent, the station starts to transmit its frame. During transmission, the station listens for a collision which would be caused by other stations attempting to use the channel simultaneously. Normally, such collisions would occur shortly after the start of a transmission because by this time all stations would have sensed carrier and deferred transmission. This interval is called the slot time and defines the maximum propagation delay through the network. If no collision is detected, the station continues with the transmission of its frame, still monitoring the channel for collisions in case of failures somewhere in the network.

As soon as a transmitting station detects a collision, transmission of the remainder of the frame is abandoned. To ensure that all stations are aware of a collision, any station detecting one invokes a collision enforcement procedure by briefly jamming the channel. Each of the competing stations involved in the collision then schedules its frame for retransmission by backing off for a random delay period. This helps to resolve contention and reduces the risk of repeated collisions. For a given transmission, the back off delay is increased each time a collision is detected until the 10th attempt. It then remains constant for a further six attempts after which, if still unsuccessful, the station will terminate transmission activity.

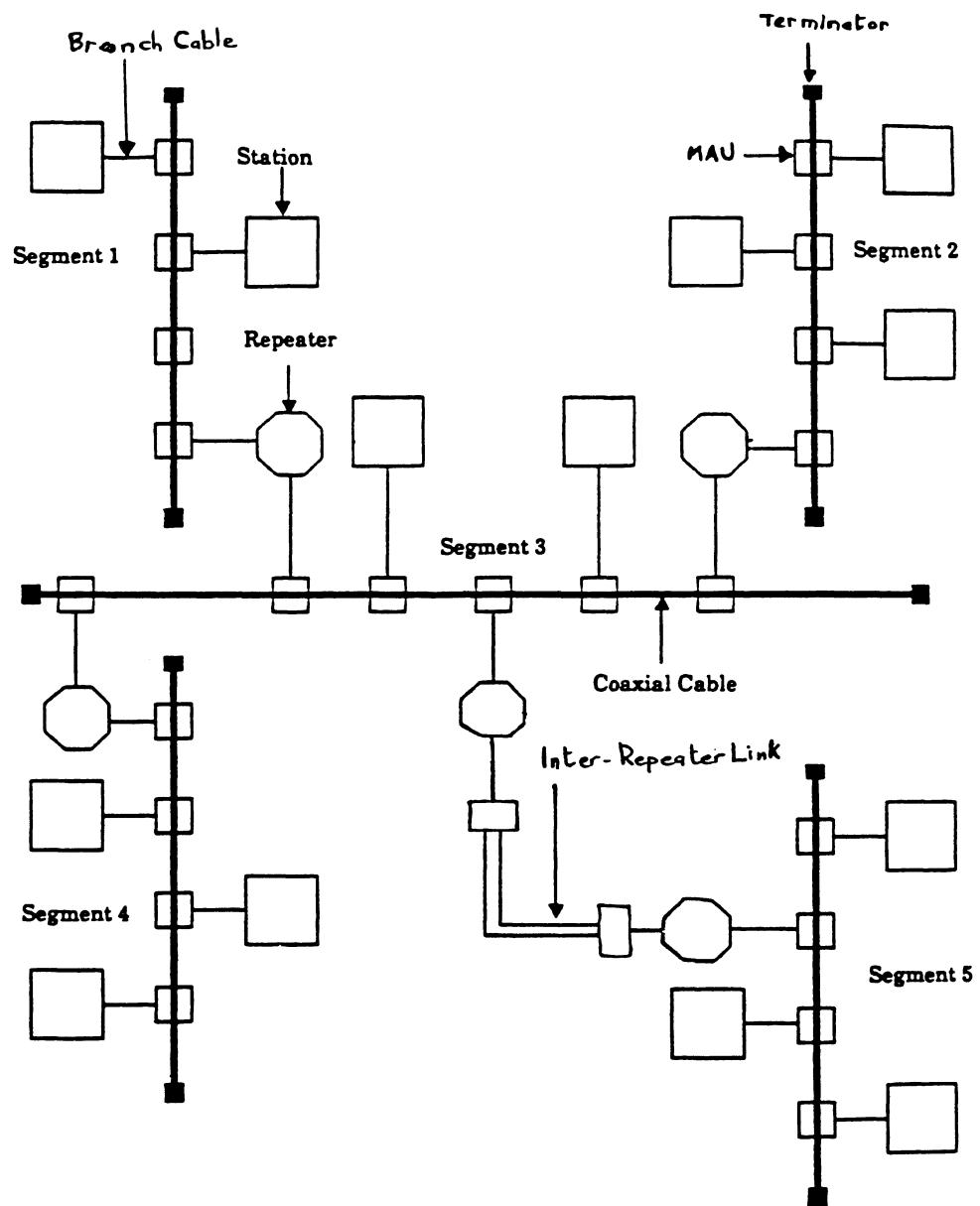


Fig. 3 Ethernet topology

Although based on comparatively simple principles, the operation of Ethernet has undergone many refinements and there are now internationally agreed standards even down to the colour of the insulation round the coaxial cable. Table 3 shows some of the characteristics of a standard Ethernet.

Figure 4 illustrates the structure of an Ethernet frame. The preamble field consists of the bit pattern 10101010 repeated seven times, allowing synchronisation of the various circuits. The start of frame delimiter is the sequence 10101011 which differs from the octets in the preamble by just the last bit. The sequence 11 therefore indicates the start of the frame proper. As shown in the diagram, the source and destination address fields are generally of length 48 bits (6 octets) though one variant of the standard allows them to be 16 bits (2 octets). The length field indicates the number of octets in the data field. If there is insufficient data to create a minimum length frame, padding octets are inserted after the data (but not included in the contents of the length field). The last field of the frame is a frame check sequence or cyclic redundancy check which protects the integrity of the frame.

A very readable account of CSMA/CD principles and standards is given in ref. 1.

6.2 The Cambridge Ring

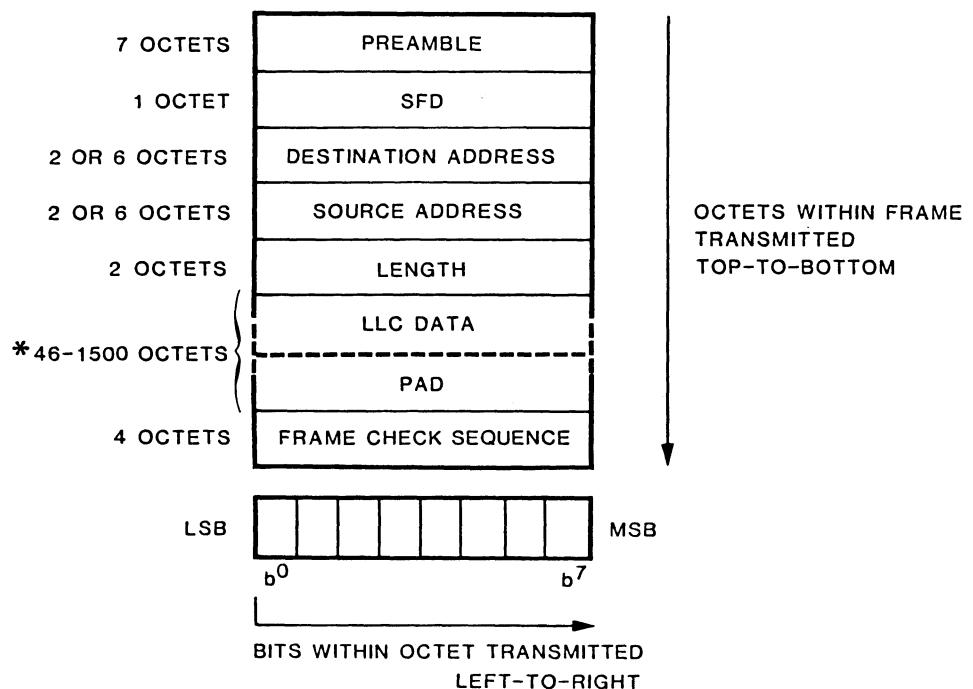
The unsurprising topology is shown in figure 5. The ring consists of two twisted pair cables and active repeaters. The delay in signal propagation through the cable and repeaters means that the system may be regarded as a continuously circulating shift register. Each 100 metres causes a delay of 450 nsec which corresponds to storage for 4.5 bits at 10 MHz. A particular configuration can contain a fixed number of bits. To impose a regular structure, the monitor station divides the bits into a number of slots, all of equal length, which travel nose to tail with a single gap of a few lengths to mark a complete cycle. The structure of the standard slot is shown in figure 6 though one variant has 38 bits (the "type" bits being omitted).

The signalling frequency is nominally 10 MHz but may vary slightly to achieve an integral number of circulating bits. To obtain an integral number of 40-bit slots plus a small gap, a variable size shift register provides additional padding.

The repeaters regenerate the signals onto each section of the ring so that rings of several kilometres can be built and different cable types can be used depending on the environments of the sections. The repeaters are also the points at which the communicating devices are attached to the network. Since they are so vital for the operation of the ring, the repeaters are powered along the ring cable independently of

Table 3
CSMA/CD characteristics

Data rate	10 Mbit/s
Slot rate	512 bit time (51.2 μ s at 10 Mbit/s)
Min. frame size	64 bytes
Max. frame size	1518 bytes
Back off delay	r slot times where r is random such that for nth attempt $0 < r < 2^k$ with $k = \min(n, 10)$
Manchester biphase	encoding



* WHEN 6 OCTET ADDRESSING IS USED

Fig. 4 Structure of an Ethernet frame

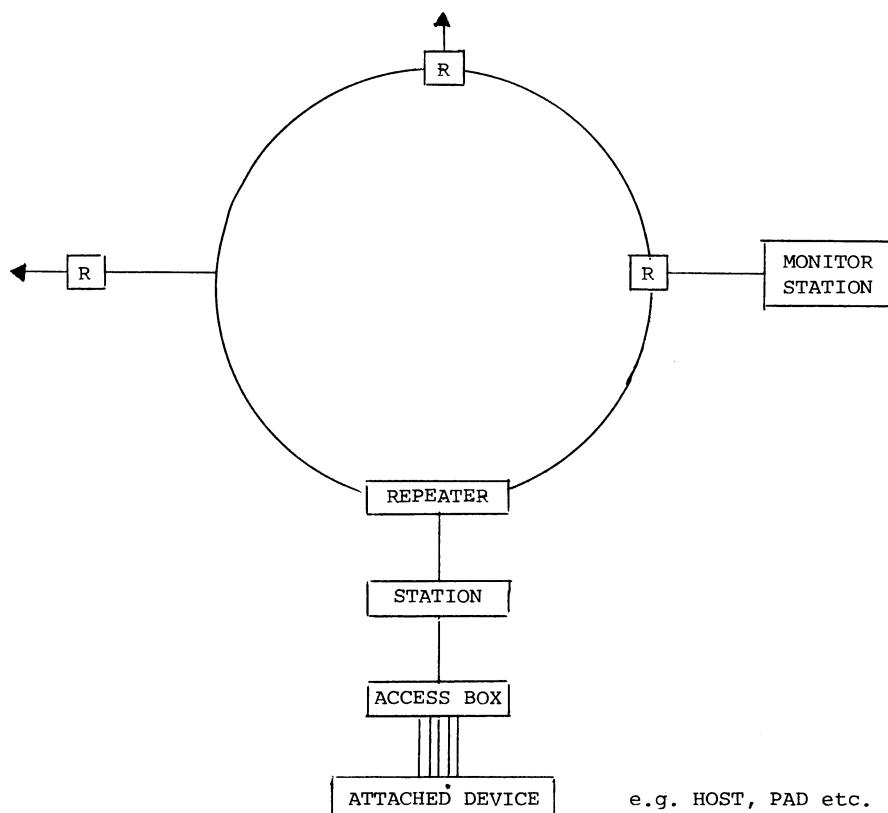


Fig. 5 Cambridge Ring topology

First onto ring

- | | |
|-------|---|
| 1 | Leader. Slot framing bit; always 1. |
| 2 | Full/empty. Set to 1 to mark slot as full. |
| 3 | Monitor pass. Set to 1 by transmitter, cleared by monitor. Prevents continuously circulating full slots. |
| 4-11 | Destination address. Identifies the node to which the minipacket is being sent. |
| 12-19 | Source address. Identifies the node which sent the minipacket. |
| 20-27 | 2nd data byte. (Data and type bits are carried transparently between interface units with user-defined meanings.) |
| 28-35 | First data byte. |
| 36 | Type bit A. |
| 37 | Type bit B. |
| 38-39 | Response. Altered by the destination node to achieve low-level flow control. |
| 40 | Even parity. |

Last onto ring

Fig. 6 Structure of a Cambridge Ring slot

the stations and attached devices.

The unit of data transmitted between stations is a minipacket which occupies exactly one slot. The first bit of each slot is set to 1 to act as the starting marker. Bit 2 indicates whether the slot is free or in use. Each station is assigned an address between 1 and 254. A station wishing to transmit waits for an empty slot then sets bit 2 to "full". It inserts the destination and source addresses into the 8-bit fields shown in figure 6, puts two bytes of data into the following two fields and initialises the response bits (bits 38 and 39) to 11. Each transmitting station can only have one minipacket in flight at a time and, after transmission, it examines each passing slot until it detects its own source address. The transmitter marks the returning slot as empty and copies the response bits. This mechanism achieves round-robin scheduling by which, however heavily loaded the ring, each slot passes from station to station giving each an opportunity to transmit within a pre-determined time interval.

Each station contains a source selector register whose value determines whether a received minipacket is to be accepted or not. If the source selector is set to 0, the station is not available to accept any minipackets while, if set to 255, it will accept any minipacket addressed to this station. For a value n ($0 < n < 255$) of the source selector, the station will only accept those minipackets addressed to this destination from source address n. The response bits are used to return low-level flow control information from destination to source.

The use of the source selector in combination with the response bits allows a receiver to listen to all transmitters and to multiplex all the incoming minipackets (if $n=255$) or to concentrate on minipackets from source address n (if $0 < n < 255$). The receiver can also generate a busy response (00) when unable to process minipackets as fast as they are being transmitted. In this case, the transmitter tries again when a new free slot passes.

As a form of frame check, the transmitter of a minipacket sets the parity bit to even parity computed over all the bits 1 to 39. This is checked by all stations. When a station detects a fault, it corrects the parity bit and sends a fault message to a logging station in the next free slot. Since the fault message contains the address n of the sending station, the logging station knows that the fault occurred in the segment before station n.

One of the stations, known as the Monitor, is responsible for establishing and maintaining the correct slot structure on the ring. When a transmitting station fills a

slot, it also sets the Monitor Pass bit to 1. This bit of a passing slot is always cleared by the Monitor station. If the Monitor detects that a slot marked full already has a zero Monitor Pass bit, an error has occurred because the slot has certainly passed its transmitting station without being marked empty. The Monitor therefore sets it to empty thereby preventing the slot from remaining permanently full and thus unusable.

The full details of Cambridge Ring standards are in ref. 2.

7. A HIERARCHY OF NETWORKS

Ideally, it should be possible for connectivity to be established between any pair of entities involved in a LEP experiment whether located on the experimental floor, in the control room, elsewhere on the CERN site or at one of the home sites of the experimenters. In practice, the best way to achieve this will be by means of a hierarchy of linked networks such as that shown as an example in figure 7. Each of the networks will be chosen on the basis of a compromise among performance criteria, available facilities and costs. Local networks, each covering no more than the few hundred metres from an experiment to its control room, may be based on technologies such as the Ethernet or Cambridge Ring. CERN is currently studying a possible wide-area network interconnecting the various parts of its enlarged site covering a diameter of about 10 kms. This corresponds to the metropolitan area of a large town. Connecting this with the outside world will be national and international X25 wide-area networks, either public or private. Each element of the hierarchy will be linked to the next by means of one or more gateways as shown in the diagram.

8. INTERCONNECTION AND INTERWORKING

8.1 Introduction

Interconnection is the term describing the ability of different systems to interwork with one another. To achieve this, it is of course first necessary to provide mechanisms for transmitting data among the systems. This is the connectivity discussed in the previous section. But, by itself, connectivity is not sufficient. In addition, all of the participating systems must interpret the data exchanged in exactly the same way.

The first packet networks were established in the late 1960's and early 1970's. The most famous is undoubtedly the Arpanet of the US Department of Defence. It currently consists of several hundred packet-switching exchanges and still serves to link users in universities, research institutions, defence establishments and industrial companies. Although a very large number of computer systems are attached to Arpanet, the different

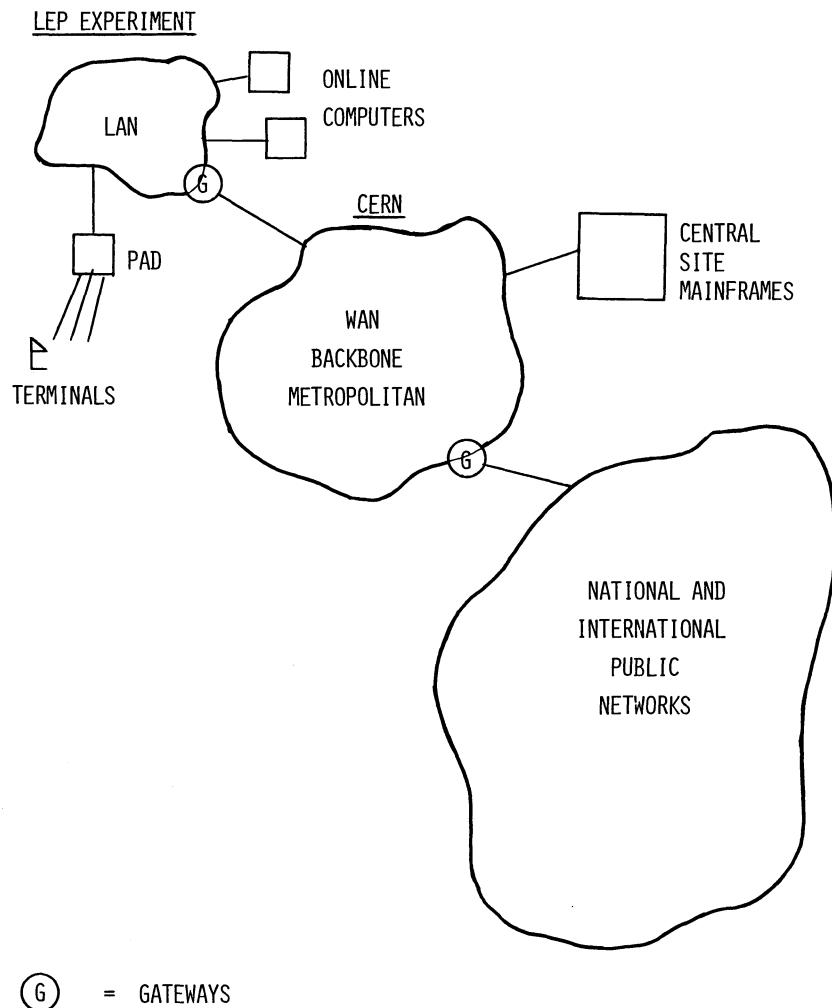


Fig. 7 A hierarchy of networks

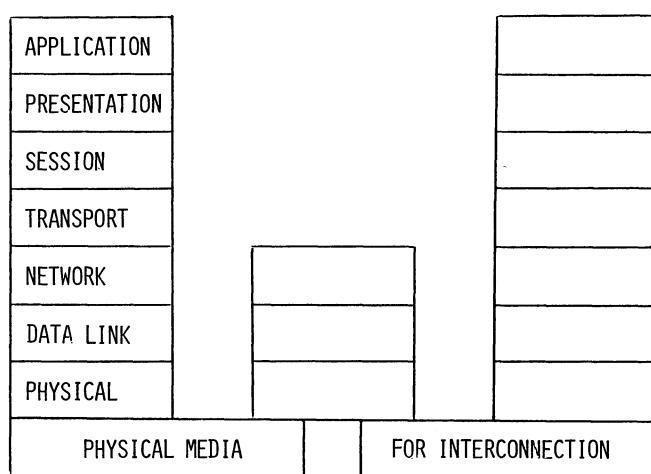


Fig. 8 The ISO model for Open Systems Interconnection

types of computer represented are relatively few. Among the main achievements of Arpanet were the work carried out on terminal access to packet networks and the use of the network for developing electronic messaging techniques. Also studied were the problems of linking dissimilar networks together (internetting) and early mechanisms for shipping files across packet networks.

In Europe, an experimental packet-switched service (EPSS) was started by the British PTT and Cyclades was set up in France. The majority of subscribers to both these networks came from the university and research communities. Characteristically for Europe, the variety of attached machines was larger than in the US. This may go some way to explaining why the research work around the European networks very quickly became focussed on the problems of interworking (i.e. interconnection) among dissimilar systems. By 1977, the British Standards Institution had been persuaded to recognise the topic as a new field of study. Similar moves were also afoot in France at around the same time. Eventually, after the usual lobbying which is prerequisite for any advancement in official standards circles, the International Organisation for Standardisation (ISO) set up a separate committee. The subject was given the grand title of Open Systems Interconnection (OSI). It quickly became evident that, if earlier standards projects in computing had been difficult, their complexity would pale into insignificance compared with the task of defining OSI standards. To render the problem more manageable, an early decision was taken to define a model thereby not only lending scientific respectability to the new discipline but also breaking it down into smaller pieces each of which could then be the subject of separate and parallel standardisation work.

The now familiar structure of the model is shown in figure 8. Among the many functions it serves are:-

i) Modularity

As with complex pieces of software or indeed any aspect of human endeavour, it is more efficient to construct the edifice from smaller building blocks each of which can be developed and checked before forming part of the larger fabric.

ii) Layer independence

By specifying clear interfaces at the boundary between two layers, the content of a given layer and the mechanisms by which it achieves its objectives can be isolated so that they become of no concern at all to other layers. This then allows particular implementations or details of a layer to be replaced as standards develop or requirements change, without impact on other layers.

iii) Emergence of standards

There is a much greater chance of reaching international agreement on a specification covering the relatively small set of functions and interfaces of a single layer than on a monolithic document covering the total OSI field. It would probably be more correct to state that the latter would be completely impossible so that the only hope of achieving international standards is by adopting a layered approach.

iv) Grouping of common facilities

There may be several different communications functions which technically sit at the same level in the model. While each of them may be independent of the others, there may be a set of common facilities which they use. It would obviously ease the task of implementors if such facilities could be grouped together within a layer for the benefit of functions at higher layers (or even in the same layer).

Apart from its role in the standardisation process, the model is also useful as a framework for the study of communications systems and as a teaching aid. However, as with all models, it must not be used indiscriminately. In other words, the model only retains its worth if it is a help rather than a hindrance in describing or understanding a particular set of phenomena. It must not be allowed to acquire the status of dogma, with reality being adjusted to fit the model rather than the other way round.

The model is a guide in deciding where functions should be placed. The standardisation process is such that the ISO definition of a layer will attempt to offer comprehensive coverage of all conceivable facilities to be embodied in that layer. In practice, difficulties are always encountered in adhering to layer boundaries when implementing real communications functions. Also, no single communications task ever requires the full set of all facilities defined in the standard for a particular layer. So we shall find that real protocols frequently cover only a subset of the facilities and furthermore that they often span layer boundaries.

Most computer manufacturers now claim to support the OSI model. Given the above remarks about the nature of the model and the relationships of real protocols to it, such statements could easily be made about almost any set of communications facilities. Many manufacturers have had their own communications architectures for some years and their statements on adherence to OSI should be treated at least with caution if not outright scepticism. On closer examination, some of these architectures turn out to show perhaps some superficial conformity to overall OSI concepts whilst being in thorough violation of

the layering details.

8.2 An Analogy

Before we become too deeply immersed in the intricacies of the OSI model, it may aid our understanding if we seek an analogy to data communications in the more familiar activity of human interconnection via the telephone system. In some places, the analogy may become somewhat stretched but the exercise may nevertheless bring out some of the basic ideas. Clearly, any characterisation of the richness and sophistication of human communications by a mere seven-layer model can only be excused if it is for purely educational purposes!

Figure 9 shows two telephones A and B, both connected to the public switched telephone network (PSTN). At the lowest level, a set of rules governs the physical behaviour of the telephone apparatus and its control of the line to the exchange. These include such details as the cable consisting of four wires and the maximum signal voltages permitted on them. The link layer of data communications has no plausible equivalent in the field of analogue telephony.

The network layer deals with the rules for making calls from one subscriber phone to another. Figure 10 illustrates the agents and the sequence of events leading to the establishment of a call and the start of conversation. The process would be slightly more complicated if one or both of A and B were attached to private automatic branch exchanges (PABX) instead of directly to the PSTN. For then, it would entail three stages - a call from A to the "gateway" between A's PABX and the PSTN, a call across the PSTN to B's PABX and a call across B's PABX to B itself. This can be regarded as a call across three interconnected networks.

So far, we have described the communications process via the telephone system. It would have been equally valid to consider what the three layers might look like using another communications medium such as, for example, a personal face-to-face conversation. The details of the lowest three layers we have discussed are dependent on the basic nature of the technology being used. The Transport Level is concerned with making the most effective use of all the underlying media available for a given communication. It also aims to minimise costs and achieve a quality of service appropriate to the nature of the application. One of its roles is to decide whether to use the telephone network (electrical transmission), a personal conversation (air vibrations) or some mixture. In practice, the transport layer is of course implemented by the human initiating the interaction. The transport level then serves to render the higher layers of the

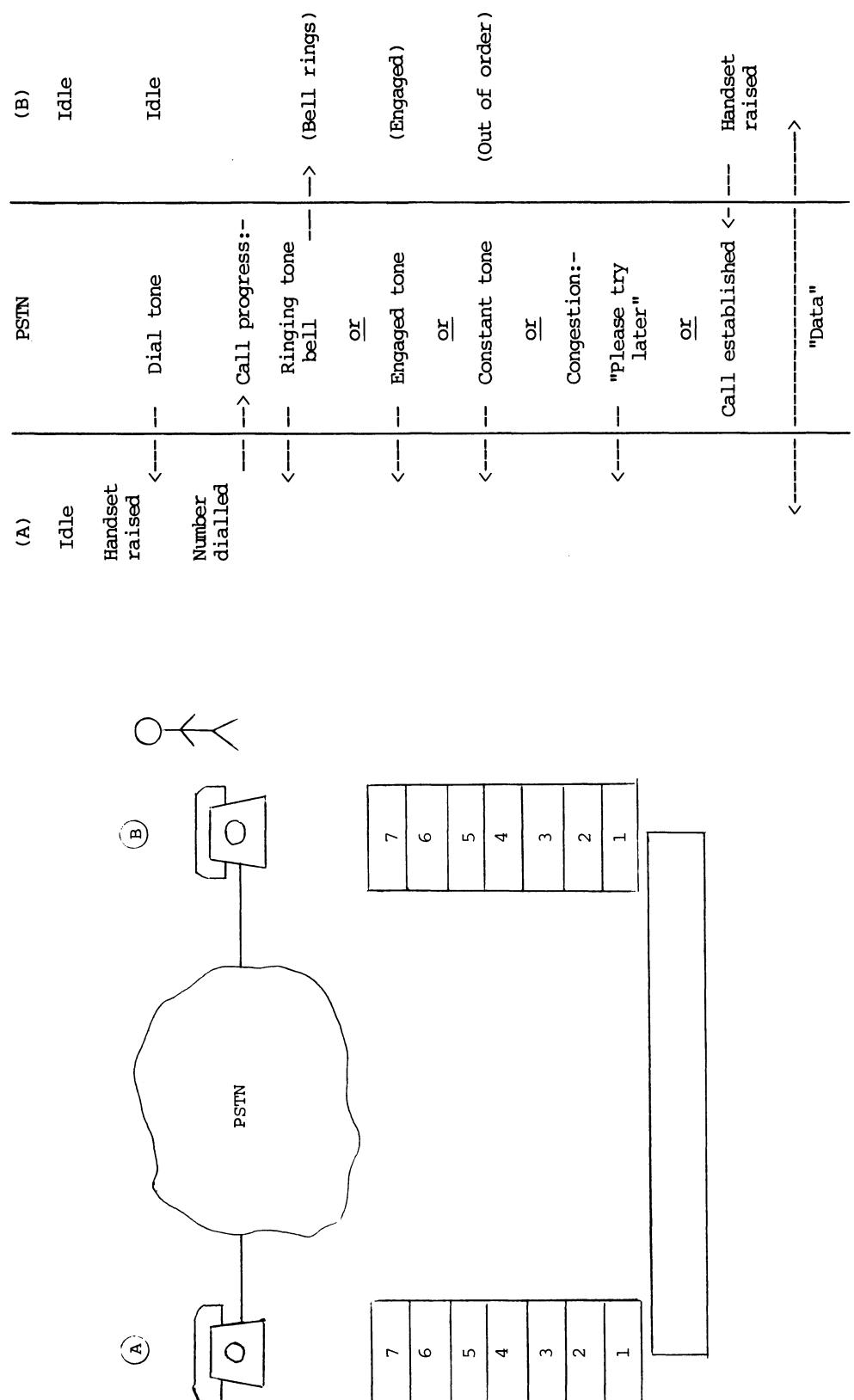


Fig. 9 7-layer model of telephone communications

Fig. 10 The network layer - making a telephone call

interaction independent of the technology used. For the purposes of our educational analogy at least, there is little difference between a face-to-face conversation and one over the phone. The upper layers are thus common whatever media are used lower down.

The session layer is concerned with the orderly conduct of dialogue. In the human case, there are implicit rules forbidding both parties in a conversation to speak at the same time and each party being given the chance to speak alternately. Of course, these are rarely obeyed but we know that the most efficient conversations are those which come closest to sticking to them. Another role of the session layer is to restore synchronisation after interruptions. When a human dialogue suffers a break at a lower level, because of a noisy telephone line or a third party entering the room, the participants have mechanisms for continuing. These entail going through a sequence of exchanges to obtain agreement on the last material to pass between them before the break. After this, they proceed in synchronism as if no break had occurred.

The presentation layer is concerned with the syntax to be used to effect the communication. In the human case, it would include the language to be used. Normally this is implicit but, in some cases such as international telephone calls, it is not unusual for conversations to begin with negotiations at the presentation level on the best language to be used for the real business of the call. The latter then takes place at the application level.

9. THE SEVEN LAYERS OF THE OSI MODEL

9.1 The Physical Layer

This is concerned with the basic parameters of the transmission process. The mechanical aspects of the layer are specifications of the plugs and sockets to be used for connecting a DTE to a communications line onto a network. The electrical aspects define the voltage levels to be used as bit values and the associated timing.

Examples of physical layer standards are V24 (CCITT) and its variants RS232C and RS449. These deal with the connection of a DTE to a modem and the interactions between them to control the physical transmission line. X21 is a similar standard for digital transmission. However, it is not pure level 1 since it includes character framing and addressing mechanisms which are elements of levels 2 and 3 in the model.

In section 5.1, it was stated that X25 is really three standards. The lowest level of X25 is defined to be either X21 for digital lines or X21 bis (which is nothing more nor less than V24) for analogue ones.

Figure 11 shows the lower OSI layers in the context of a DTE on a local area network while figure 12 indicates the mechanical and electrical details for the physical layer of a CSMA/CD connection. Manchester Biphase encoding combines clocking signals and data into bit symbols. Each bit symbol is divided into two halves, the second of which contains the binary inverse of the first. A transition always occurs in the centre of each bit symbol. This sequence of transition forms a square wave whose frequency is half the clock rate. Thus the bit rate of a 10 MHz Ethernet is 5 MHz.

9.2 The Data Link Layer

Whatever the nature of a network, it will consist of a number of links each of which will need to be controlled by the entities at either end of it. This is the task of the Data Link Layer. The various functions of data link protocols are discussed below (see ref. 4 pp 1378-1383). Whether and how a function is performed vary with the type and sophistication of the protocol used.

i) Initialisation

The initialisation function deals with the establishment of an active Data Link connection over an already existing physical path. The physical path may be built on one or more physical circuits. The acquisition of the path and the movement of bits over the path are the responsibility of the underlying physical layer processes. Initialisation usually involves the exchange of supervisory sequences establishing readiness to receive or transmit and, if necessary, identification of the parties.

ii) Synchronisation

The underlying Physical Layer provides a stream of synchronised bits. A function of the Data Link Layer is to determine where in this stream of bits the intelligence being transferred lies. The synchronisation process accomplishes this by providing functions to acquire, maintain and, if necessary, reestablish character synchronisation, that is, bringing the receiver's decoding mechanisms into alignment with the transmitter's encoding mechanisms.

iii) Transparency

The transparency function permits the Data Link control to be totally "transparent" to the format or structure of the users' information. Transparency permits the user to send information in any code set, of any length and in any format with the assurance that Data Link mechanisms will not "trip over it", that is, will not interpret any user data as link control information. Data Link protocols vary widely in the techniques used to provide transparency.

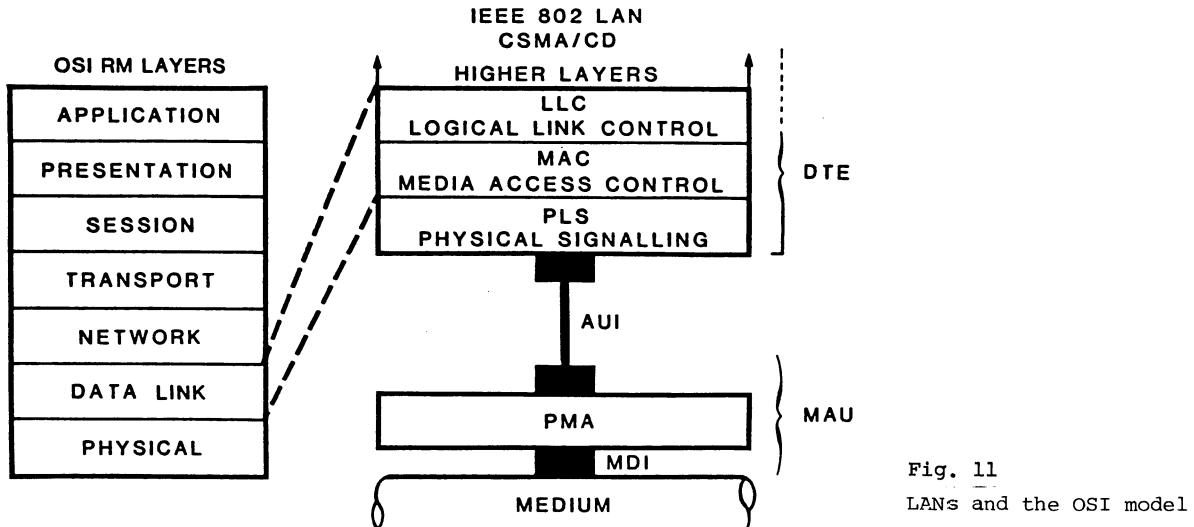
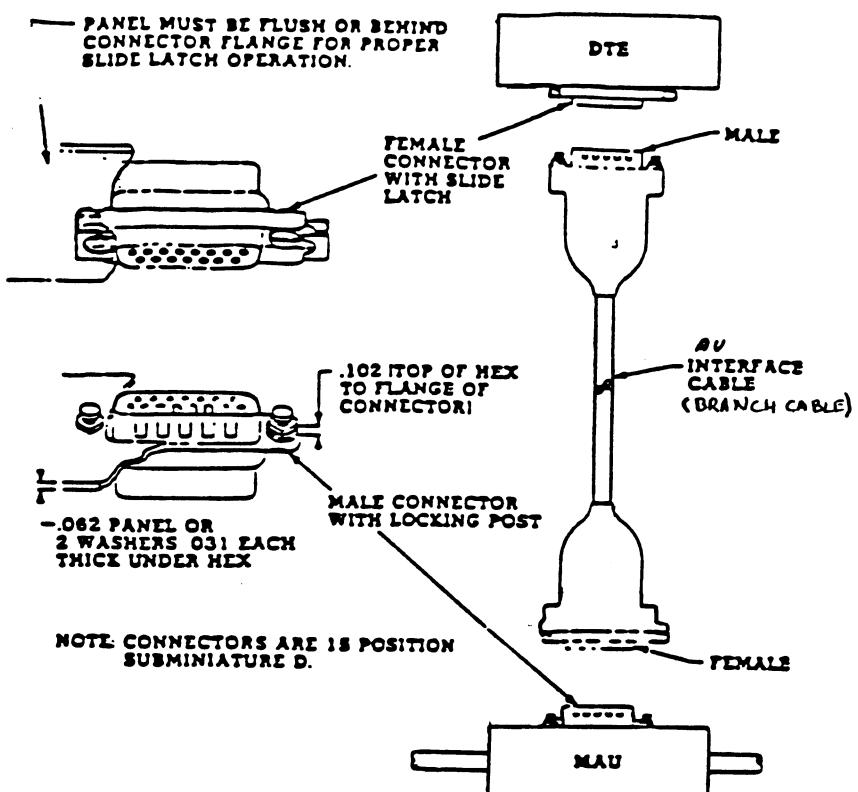


Fig. 11
LANs and the OSI model

MECHANICAL



ELECTRICAL

- MANCHESTER BIPHASE ENCODING

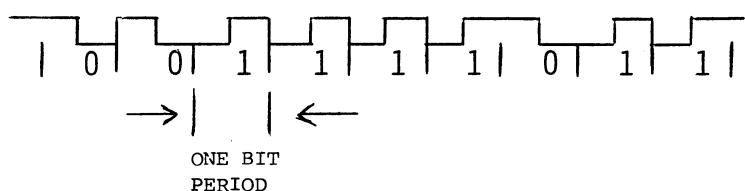


Fig. 12 The physical layer of Ethernet

iv) Segmentation

A blocking or framing mechanism is necessary to divide the users' information into segments suitable for transmission through the Data Link. Extremely long blocks of information are unlikely to survive transmission through a noisy medium without error. On the other hand, very short blocks may be inefficient. Blocking and framing mechanisms aid the synchronisation process and provide the ability to identify when data should, but may not, be present. They also provide convenient segments on which to apply error-detection processes.

v) Flow control

Receivers need to be able to regulate the flow of information into their systems in order to prevent being overwhelmed by incoming data in cases where the input rate exceeds the station's capacity to accept and process the data. Flow control functions accomplish this regulation.

vi) Error and sequence control

Error control processes provide for the detection of errors induced by the transmission medium, the acknowledgment of correctly received segments and requests for retransmission of segments containing errors. Vertical, longitudinal and cyclic redundancy checks are the most commonly used error-detection techniques. Some Data Link protocols also employ sequence control, which numbers and verifies individual segments of data, to guarantee the detection of missing segments.

vii) Recovery

This function includes the processes required to detect and recover from abnormal occurrences such as loss of response, illegal or invalid sequences, severed links and the many other unpleasant things that can happen to information as it moves over the link. Timeout processes are a common method of detecting such occurrences.

viii) Termination

Following the transfer of the users' information, the link, which was logically established by the initialisation process, is terminated. Termination functions involve "tidying up" the link by ensuring that all data sent has been received and then gracefully clearing the logical connection. Link termination does not necessarily involve disconnection of the physical path.

Early data link protocols were character- or byte-oriented. Each frame consisted of characters in a particular code (e.g. ASCII or EBCDIC) with the consequence firstly that communications between computers with different character lengths was difficult and

secondly that there had to be a special mode to achieve data transparency. IBM's BSC is an example of such a code.

The disadvantages of embedding character codes in the design of data link protocols led to the emergence of so called bit-oriented protocols whereby data frames can contain an arbitrary number of bits and frame sizes need not be integral multiples of a particular character size. There are a number of bit-oriented protocols of which IBM's SDLC (Synchronous Data Link Control) was among the earliest. Variants of this include ANSI ADCCP (Advanced Data Communication Control Procedure) and ISO HDLC (High Level Data Link Control).

Level 2 of X25 is defined to be the so-called LAPB option of HDLC. This means, among other things, that a DTE's data link to a packet switched network is controlled according to the HDLC rules. Because of its importance for X25 and because it exhibits features common to all bit-oriented protocols, HDLC is worth a bit more attention. Figure 13 shows the structure of an HDLC frame. Without going into too much detail, let us see roughly how various of the above functions i) to viii) are carried out. The control field of a frame specifies its type and initialisation is signalled by one end sending a special control frame. Synchronisation is the means of indicating frame boundaries. For this purpose, the first and last octets of a frame each consist of the bit sequence 01111110 forming flags. A second role of the flags is to segment the data into convenient units for the transmission process. Transparency is needed to ensure that the user's data can contain any sequence of bits, even an HDLC flag if he feels like it. To avoid such data causing chaos by being treated as a flag delimiting a frame, a technique known as bit-stuffing is used. Before being put onto the transmission line, data between the flags of a frame is scanned by hardware, which inserts a 0 after any sequence of five consecutive 1's. The reverse process takes place at the receiving end.

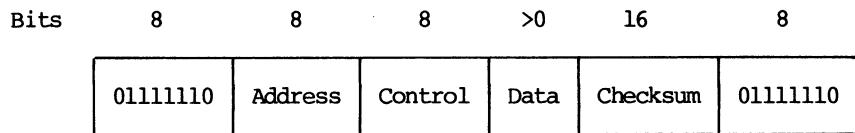


Fig. 13 The structure of an HDLC frame

Flow and sequence control and acknowledgment are performed using sequence numbers in precisely the same way as described for X25 level 3 in section 5.1. However, it is important to note that, although the principle is identical, the sequence number of a level 2 HDLC frame is not to be confused with that of an X25 level 3 packet which it might be carrying. For error control, a frame check sequence is computed by means of a cyclic redundancy formula and embedded in the 16-bit field of the frame. To terminate a connection, a DISCONNECT control frame is transmitted.

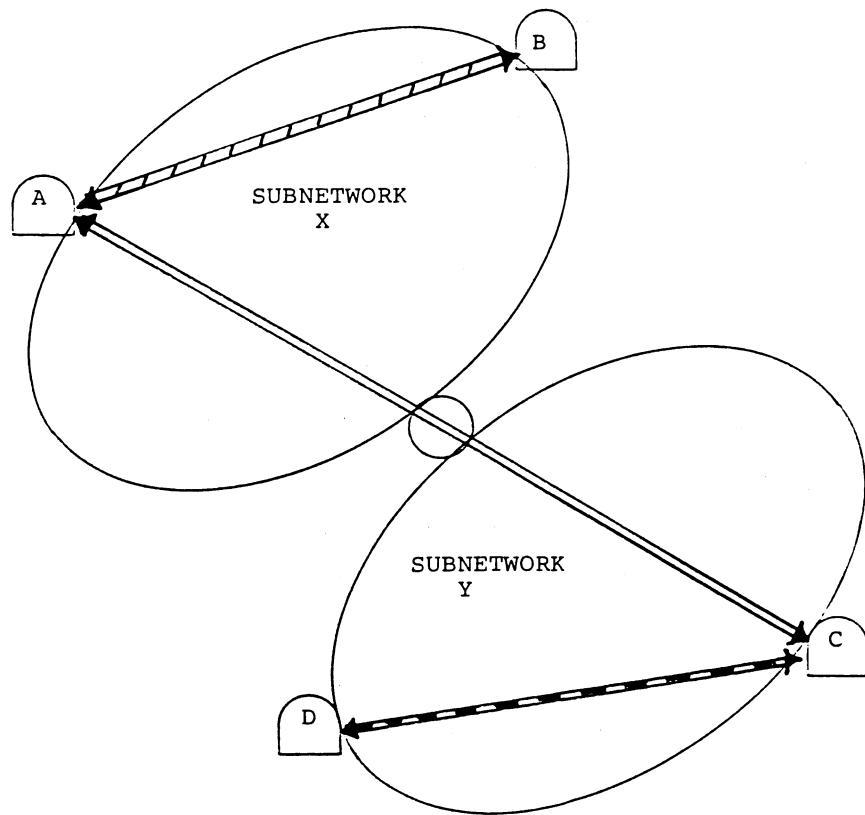
An implementation of the data link layer on a CSMA/CD local network can be achieved by embedding HDLC frames in the data field of an Ethernet frame. (Care should be taken here to distinguish the use of the term "frame" in two different contexts.) In this case, the Ethernet could be regarded as a point-to-point HDLC link between the communicating parties. The procedure is known as Logical Link Control type 2 or LLC2.

9.3 The Network Layer

Data communications can be viewed as the passing of information between "processes", residing in systems which could be separated by a complicated set of interconnected networks of different types. The function of the Network Layer is to provide a pathway between the processes along which the data can travel. Figure 14 shows two subnetworks X and Y of different types connected by a gateway. Subscribers A and B on X can communicate according to the rules of X so can C and D according to the rules of Y. What about A and C? If we remember our telephone analogy, in which the subscribers were separated by the PSTN plus their local and remote PABX's, the gateway at each PABX was able to match the local PABX conventions to those of the PSTN.

It is the task of the Network Service to provide such a match by bringing the often disparate services in the subnetworks up to a common level. One way of doing this is illustrated in figure 15 which shows different network services AF1 and AF2 in two subnetworks 1 and 2. The two end systems wishing to interwork have to do so by means of enhancement functions EF1 and EF2 which raise the respective network services to the level of the common Network Service. Of course, end systems on the same subnetwork do not have to use enhancement functions but then they would require different network services for internal and external communications.

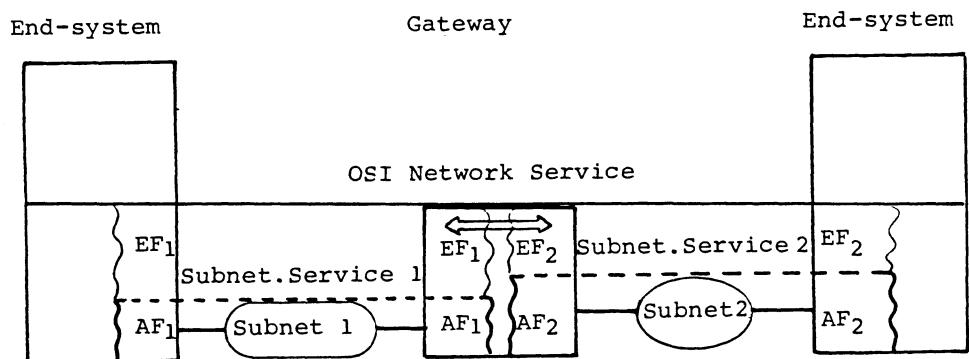
A particularly troublesome and long-standing difficulty in networking is the issue of addressing. This is not the place for a lengthy explanation. Suffice it to say that the network layer is where the addresses of the end processes make their first appearance in the model.



— Communication using service provided
by subnetwork X

— Communication using service provided
by subnetwork Y

Fig. 14 The interworking problem



AF_i = access functions for subnetwork i

EF_i = enhancement functions for subnetwork i

Fig. 15 Hop-by-hop enhancement for the Network Service

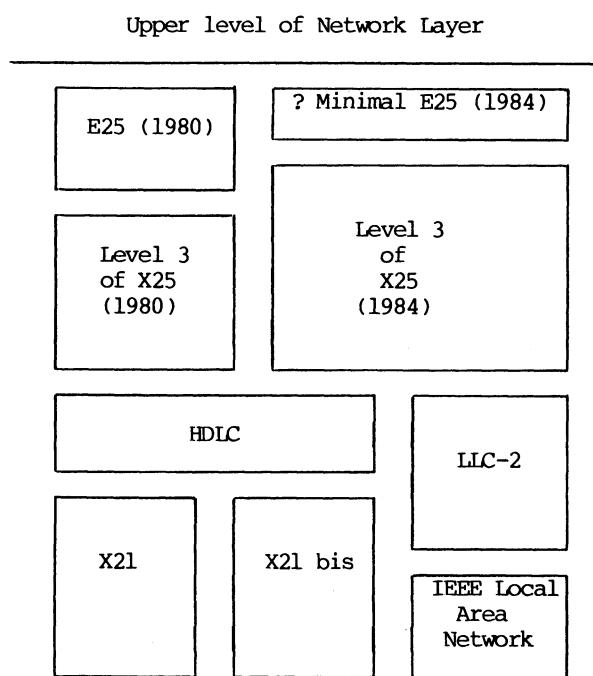


Fig. 16 Implementations of the Network Service

Figure 16 illustrates how the ISO Network Service might be implemented over a number of different types of subnetwork.

A popular misconception is that X25 level 3 offers a Network Service in the ISO OSI sense. In the 1980 version of X25, a mere 12 decimal digits were allocated for addresses. A subscriber to a data network was viewed as if it were a telephone only capable of sustaining one call at a time. In practice, computer systems can run large numbers of processes simultaneously and each could be involved in an independent data communications activity. Hence the need for a much more flexible means of assigning addresses to processes. Therefore, if X25 (1980) is used in a particular subnetwork, some enhancement of the addressing mechanism is needed for connections into other subnetworks. This is shown as E25 in figure 16. The 1984 version of X25 offers some improvement with a mandatory address length of 32 decimal digits. There may also be eventual agreement on an optional extra portion of up to 512 bits. In this case, the enhancement function E25 (1984) would be even smaller than E25 (1980).

For a local network in which LLC2 is used as the Data Link layer, X25 level 3 could, with appropriate enhancements, form the basis of the Network Layer, as in figure 16.

A useful discussion of the Network Service and implementation recommendations are contained in ref. 3.

9.4 The Transport Layer

For a pair of processes in remote systems between which communications are to take place, there may be various Network Services available from which to choose. For example, the systems might be attached both to the PSTN and to packet-switched networks. Each Network Service will offer a particular Quality of Service (QOS) which will be highly dependent on the technology of the underlying communications mechanisms. Among the elements of the QOS are cost, throughput and residual error rate. The Transport Service is there to optimise the use of whatever Network Services are provided, taking account of the technical performance and economic requirements of the end processes. Included among its functions is the addition of capabilities to overcome whatever inadequacies might exist in the Network Service which might otherwise prevent the process-to-process connection from taking place at all. As an example, some inter-process communications might demand a lower error rate than could be provided by the Network Layer, in which case the Transport Layer would have to include additional error detection and correction mechanisms.

Another example is the use of the Transport Layer to reduce costs. All public data

networks include the duration of a call as an element of its cost. If there are two systems between which a large number of simultaneous and independent calls are taking place, it may be possible to establish a single Network Service connection and to multiplex along this the traffic for all the calls. This might result in more efficient use of the connection and lower duration charges than if a separate Network Service connection were established for each call. "Tariff-busting" of this kind is the role of the Transport Layer.

Since the Transport Layer has the responsibility of managing the exploitation of the Network Layer, all the higher layers are independent of the underlying communications technologies.

9.5 The Session and Presentation Layers

In general, the layers discussed so far are concerned with the provision of general purpose communications resources which can be shared among a number of end processes and their use optimised for specific purposes. The remaining layers define how these mechanisms are to be used and the boundaries between layers are less distinct. Layers 5 and 6 (Session and Presentation) group together various facilities common to several if not all applications.

In Layer 5 are embodied the tools for managing dialogues (or "sessions") between end processes. This is achieved by ensuring the orderly exchange of control and data blocks, for example using a token passed, via the session layer, by the end processes. Only the process with the token may transmit. This is also the level at which recovery from traumatic problems at lower levels in the hierarchy can take place. The session layer can insert synchronisation marks into a stream of data between two processes and require the acknowledgment of each before continuing with the data transfer. The advantage of this mechanism can be appreciated by considering the transfer of a very large file which is interrupted, shortly before the end of the file, by a network error of some kind. Instead of having to begin again, each of the processes is aware of the last synchronisation mark acknowledged so transmission can recommence from that point.

There are circumstances when it would be dangerous to carry out some action in advance of the completion of a particular communications activity. For example, the updating of a database could involve a lengthy exchange between remote processes. To carry out partial updates while the connection is open might lead to corruption of the database in the event of a network error. The Session Layer implements facilities for quarantining data during the transfer phase. This means that no action apart from data

receipt or transmission is permitted by either of the end processes until the transfer has been completed.

The presentation layer deals with the syntax of the data passed between the application layer processes. One of its functions is to handle the inevitably different conventions for representing information used by computer systems from different manufacturers. Obviously, there has to be some agreement on the codes to be used for a transfer. The presentation layer allows negotiation of the syntax; an example being whether characters are to be encoded in ASCII or EBCDIC. A further function of this layer is to improve the efficiency of transfers by using data compression techniques. One of these is to reduce the length of strings containing sequences of identical characters.

9.7 The Application Layer

The seventh layer is concerned with the abstract specification of some activity to be performed for which communications mechanisms are indispensable. In other words, the activity is distributed among a number of physically separate systems. Examples include the transfers of files and jobs across networks. The standardisation of the application layer is more complex than the lower layers. Not only must it deal with communications-oriented services but it must also cover relationships between the applications and the real world.

The activities of file transfer, access and management (referred to as FTAM in ISO circles) require definitions of a file store and the files in it both in terms of their properties and their structure. This is achieved by means of a virtual filestore whose elements can be referred to by the communications part of the standard. It is then the task of implementors to map the virtual file store onto real system components. These need not be restricted to real file stores but could also include remote peripherals on electronic mailboxes.

Like FTAM, job transfer and manipulation (or JTM for short) also entail the definition of general concepts with roles similar to those of a virtual file store. It must be emphasised that JTM does not pretend to tackle the problem of job control but, initially at least, only provides facilities for transferring "work units". Each work unit contains information and instructions on what is to be done with it at various points on its journey. For example, a work unit could contain a job with an indication that it is to be submitted to some job mill. This in turn could generate output in another work unit to be sent to a printer somewhere on the network. All the steps could result in progress reports (yet more work units) being sent to a nominated collection

point at which a history file of the job could be built up. Additional functionality (and consequent protocol complexity) could be introduced by allowing jobs to be spawned by other jobs and the transmission of work units containing requests to change or interrogate other work units. This last feature will provide the users with facilities for job visibility which will be essential to maintain track of jobs progressing through various systems in a network environment. It is possible that agreement will be reached on the use of ISO FTAM as the bulk transfer mechanism for carrying JTM work units - an interesting example of a service being offered to an entity in the same layer.

10. THE STATUS OF OSI STANDARDISATION

Standardisation requires agreement to be reached among a large number of parties with very different perspectives. The process can therefore become protracted. The last few years have seen feverish activity on OSI standards within the various national and international organisations. The first objective is to get a set of definitions agreed within ISO and then to ensure that the suppliers of information technology products implement them.

Each layer in the model performs a set of functions or services which it offers to the layers above. In turn, it makes use of services from the layers below. There are therefore three main elements to a standard for an entity within a particular layer:-

- i) the services and interfaces it offers to the layers above
- ii) the services it requires of the layers below and the lower interfaces
- iii) the rules (or "protocol") for exchanging information with a peer entity across a series of networks.

No attempt is made to indicate how an entity should be implemented. This is not amenable to standardisation because it depends heavily on such things as machine architecture and operating system design.

Elements i) and ii) above are normally combined together in a "service specification" while iii) constitutes a "protocol specification". Each of the specifications can be worked on simultaneously by the relevant ISO groups. A standard evolves in three stages - draft proposal (DP), draft international standard (DIS), full international standard (IS). The status of standards for the various layers and functions described is shown in table 4.

Inevitably, the standards will emerge piece by piece over the next few years. Indeed, given the complexity of the task, it would be a remarkable achievement if the

Table 4
Status of ISO standards for OSI

	1982	1983	1984	1985	1986	1987
Files			DP DIS	IS		
Jobs			DP	DIS	IS	
Presentation			DP	DIS	IS	
Session		DP DIS	IS			
Transport	DP	DIS	IS			
Network		DP	DIS	IS		

DP = Draft Proposal
DIS = Draft International Standard
IS = International Standard

timescales could be met. Assuming that the present enthusiasm and energy can be maintained, there is the prospect of a complete basic set of ISO standards for OSI by 1987 and of corresponding products becoming available commercially before the end of the decade.

Refs. 4 and 5 contain further details of progress on standardisation.

11. PRACTICAL PROTOCOLS - OSI IN ACTION

11.1 Background

It might appear that the idea of implementing an ambitious scheme based on the principles of Open Systems Interconnection would be nothing more than a computer scientist's pipedream. To show that, with the right combination of circumstances, even the seemingly impossible is possible, we focus on the United Kingdom's university and research community. Here a national project has been in progress to achieve widespread interworking among about 150 institutions. Each site generally has its own local computing resource and there are, in addition, a handful of national facilities. The computers are of all shapes and sizes, from a large number of different manufacturers. The financing for computing in this environment is centralised and the funding body has a policy of resource-sharing and site-specialisation. As an example, two of the national centres have supercomputers which users at all sites are entitled to use, subject to the allocation of the appropriate resources. Wide-area data communications are therefore an immediate prerequisite for the implementation of the central policy.

In 1976, a study was begun into the best way of achieving the desired connectivity and interconnection mechanisms. The resulting recommendations were adopted and a centrally coordinated project was established in 1979. By this time, each site had not only its wide-area but also its local communications needs and these became an equally important aspect of the activities.

In the following sections, we consider the practical solutions adopted for the UK project to implement the various facilities discussed earlier.

11.2 The Wide Area Network

By the mid-1970's, a small-scale private packet-switched network had been established for the UK research community. This used the access protocols defined for the EPSS network mentioned in section 8.1. With the ratification of X25 in 1976, the private network was converted to conform with the new standard and it expanded until it eventually had subscribers at academic sites in most parts of the UK. By this stage, the

public X25 service PSS had been opened by the British PTT. There were then three main classes of communications arrangements: the private X25 network for research workers, the public X25 PSS network and a number of star RJE networks onto the national university computer centres. One of the objectives of the national project was to unify these disparate facilities. The main decision was whether to use PSS or to expand the private network so that it encompassed the whole of the university community not just selected research departments.

In section 5.3, it was stated that the tariffs for public X25 networks include a significant element based on volumes of data transmitted. In the UK academic community, a considerable proportion of the traffic is related to the transfer of jobs, files and output. In general, public network tariffs favour interactive usage with comparatively low volumes of traffic generated. PSS charges for the community's traffic profile and volumes would have been very high. In addition, major capital investments had already been made in the private network. These two factors were the principal reasons why it was decided to base the unification on the further extension of the private network (which was named the Joint Academic Network - JANET). PSS would be used via two gateways to link the community with the outside world.

11.3 Local Area Networks

There were two schools of thought on the technology of local area networks. One believed that, since networks of 10 Mbit/s could be easily and cheaply constructed, solutions based on techniques such as the Cambridge Ring or Ethernet were the only sensible choice. The other said that the high speed solutions should be viewed with caution because no standards had yet been defined for them. As an interim measure, X25-based LAN's, whilst offering comparatively low performance, would at least offer ports for the attachment of subscriber systems in conformity with international standards. In the event, both types of approach were adopted. Since the Cambridge Ring had emanated from a UK university, this high speed technology received considerable attention, with the aim of having components in the marketplace from different suppliers. As soon as an operational requirement had been produced, it became apparent that all the potential manufacturers would offer incompatible products unless some standards could be agreed upon. An unprecedented exercise then took place in which customers, a number of competing suppliers and the Department of Industry all collaborated to define Cambridge Ring standards of the kind described in section 6.2. Allowing for the normal gestation period, products were then developed and customers could obtain components from several sources.

Meanwhile, the LAN committee of the US Institute of Electrical and Electronic Engineers (IEEE) was working hard on a range of standards including CSMA/CD. Several computer manufacturers had declared their intentions to adopt these for their LAN products. Ethernet therefore became irresistible as one of the technologies for LAN's in the UK academic community. The present position is that, while a few universities have Cambridge Ring LAN's, there is a discernible swing towards Ethernets as the de facto standard solution.

For low-speed X25 LAN's, a manufacturer was found who could offer a single node packet-switched network with ports running at a maximum of 48 kbit/s. This solution found favour in about 40 universities especially where an off-the-shelf product was required which would not need extensive manpower to bring it into service.

Early in the life of the project, a need was identified for cheap PAD's of the kind described in section 4.2. In the absence of a suitable commercially available product, a design was undertaken based on developments already taking place in one of the universities. Approaches were made to several potential manufacturers and one of them was chosen to collaborate in a joint venture. The resulting product is the so-called JNT PAD of which over 1000 have been installed throughout the community and beyond.

The first network interface developed for the PAD was X25 and it was designed to handle synchronous ports for host computers at up to 9.6 kbit/s as well as asynchronous ports for terminals. The PAD is also capable of low-speed switching and of acting as a reverse PAD. Figure 17 shows examples of PAD configurations. More recently, Ring and Ethernet interfaces have been developed for the PAD. Thus the same basic equipment and user interface can be used with a variety of LAN technologies.

11.4 Connectivity

Figure 18 shows the hierarchy of local and wide-area networks to achieve connectivity in the academic community.

11.5 Protocols

The UK project was already well under way by the time the ISO activities started. From the studies described in section 8.1 emerged an architecture in which elements of the ISO model were already clearly discernible. The subsequent protocols became known as the "Rainbow" Books because each was bound in a different colour. Figure 19 compares the two architectures.

The crucial interface is that offered by YBTS. This is the same whatever the underlying communications media and is obtained by appropriate enhancements to the lower

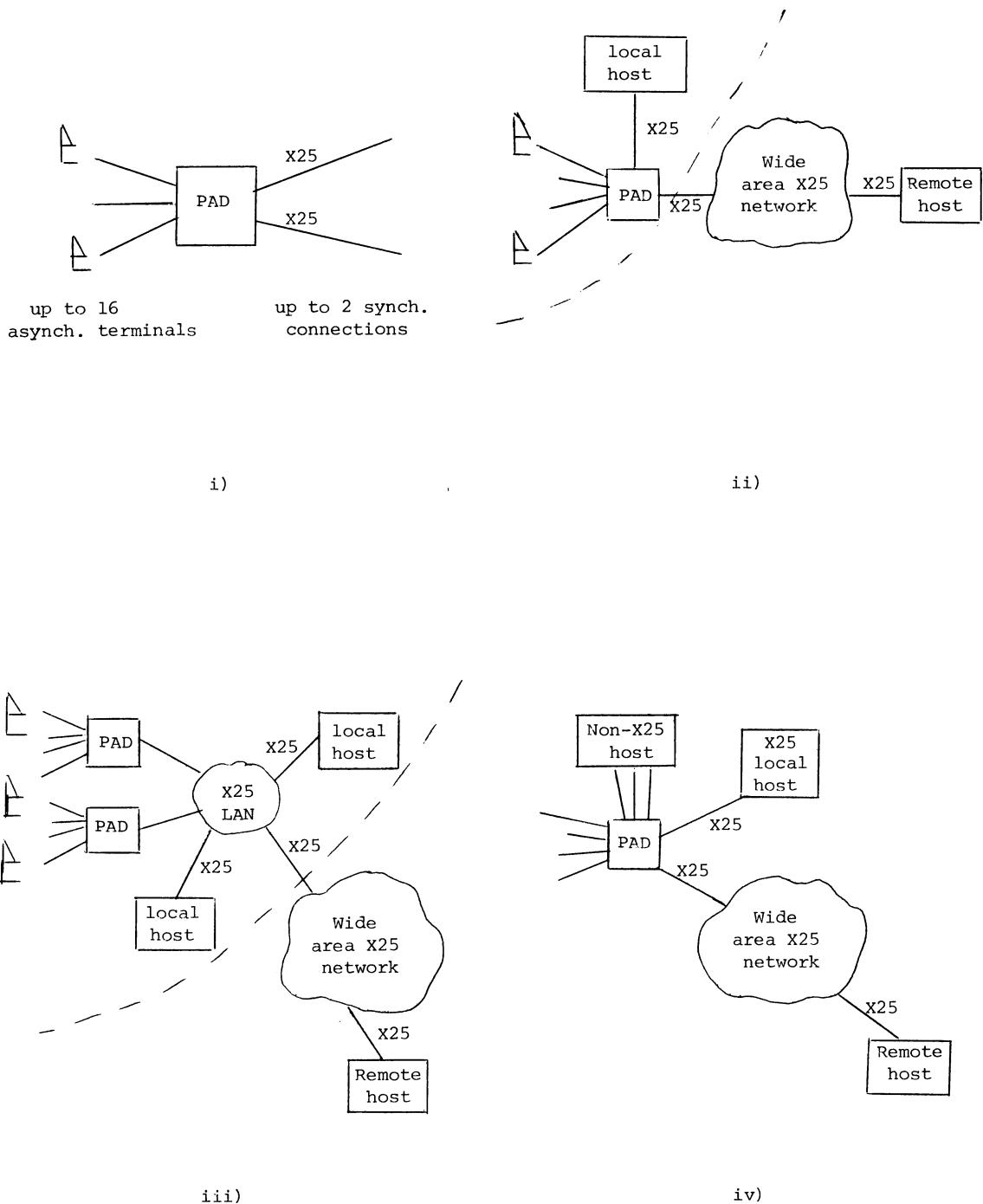


Fig. 17 PAD configurations

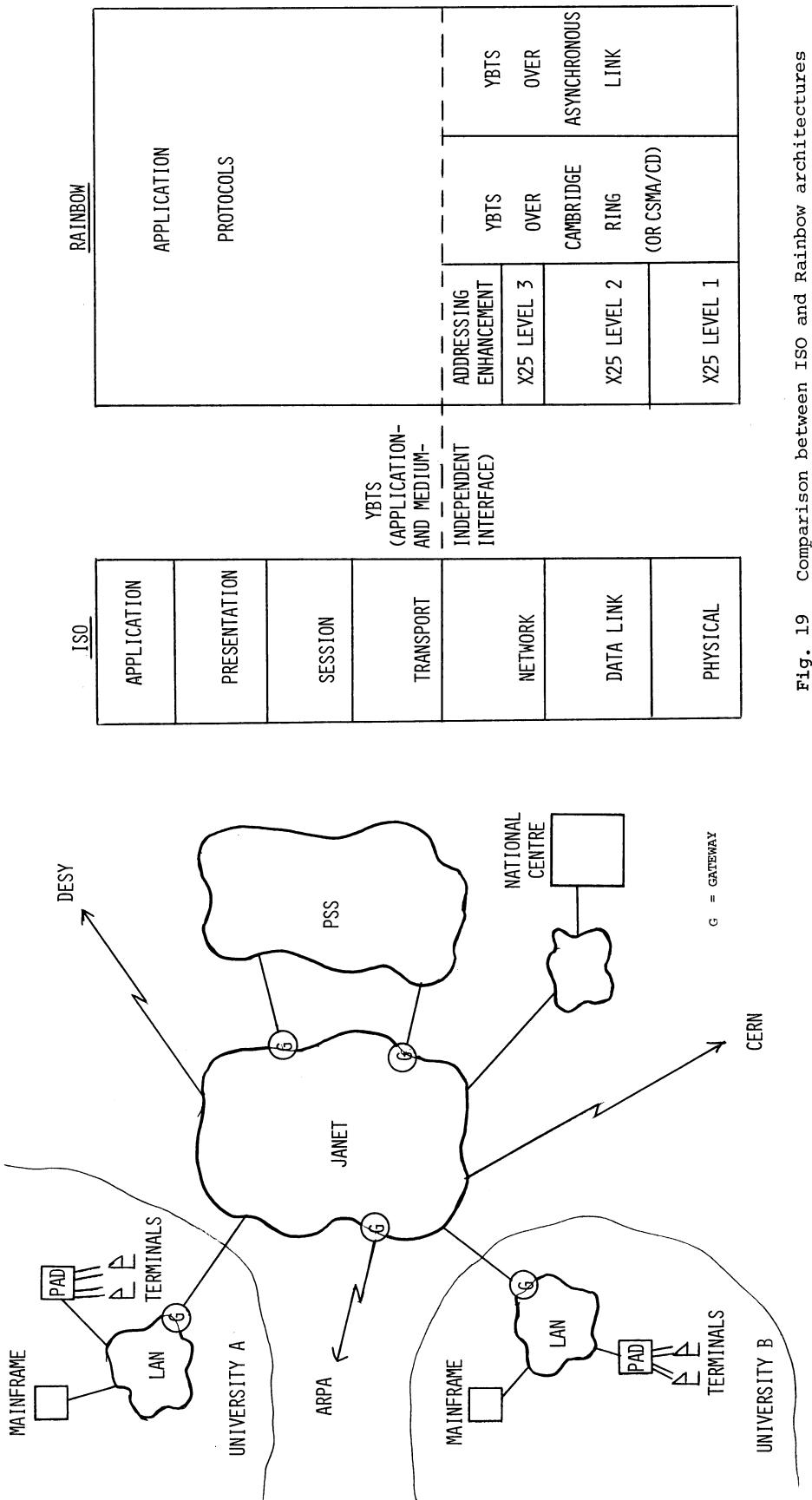


Fig. 18 Network hierarchy for UK academic community

Fig. 19 Comparison between ISO and Rainbow architectures

layers just like the enhancement functions of figure 16. YBTS is therefore just a Network Service interface for use by the upper layers. (Unfortunately, the term YBTS which stands for Yellow Book Transport Service was coined before the ISO terminology came into being. This is the source of much confusion.)

Instead of separately identifiable protocols for the session and presentation layers, these are embedded in the Rainbow application layer protocols thereby making them somewhat more monolithic and complicated than necessary. The Blue Book File Transfer Protocol (FTP) provides for the general movement of documents. It includes recovery mechanisms (corresponding to ISO level 5) and the negotiation of character representations (ISO level 6). The Grey Book Mail Protocol moves files between mail facilities using the above FTP as the bulk shipment mechanism. It can also be used for relaying mail via third parties. The Red Book Job Transfer and Manipulation Protocol (JIMP) handles the movement of descriptors which correspond to the ISO work units - again using FTP. The different types of descriptor allow jobs to be queued for a job mill, outputs to be transferred, reports of job-related activities to be collected, the status of job-related queues to be interrogated or the status of individual jobs on a queue to be modified.

Ref. 6 is a summary of the architecture and protocols while refs. 7 and 8 contain the detailed specifications of FTP and JIMP.

11.6 Protocol Implementation

An early objective of the project was to obtain maximum support for the chosen standards from manufacturers. Since the definitions had no status within the official standards bodies, it was not expected to be easy to win manufacturers' cooperation. One approach was to include some form of support for the Rainbow Books as a mandatory requirement in the procurement of new equipment. This proved to be partially successful in getting manufacturers interested in collaborative ventures. The reason for this sympathetic attitude was that, although not a large and lucrative market, the academic sector is well publicised and influential. It is also recognised as a useful test-bed for important new ideas such as OSI.

However, it would have been totally unrealistic to expect fully supported implementations of all the Rainbow protocols to arrive from the manufacturers. Therefore, a limited number of development projects were funded in the universities, each concerned with implementing a particular protocol under one operating system. In this way, the full protocol repertoire was gradually built up for most of the more common computer systems.

11.7 Present Project Status

The hierarchy of figure 18 is essentially complete with nearly all university sites now connected via their IAN's to the national network. Many computer centres have mounted implementations of the Rainbow protocols which are already forming the basis of many of their services. As an example, the author's own centre will by mid-1985 be offering remote job entry services based exclusively on the use of JTIP.

11.8 The Future

Technological pioneers always pay a price for being innovators. In this case, the use of pre-ISO protocols will mean a major transition when the formal standards emerge. Nevertheless, the importance of the present project must not be underestimated for it has been responsible for creating an unrivalled reservoir of expertise on networking techniques and architectures, for providing direct operational experience of an Open Systems environment and for developing appropriate hooks in several common operating systems. These last are independent of the particular protocols used and will be as valid for ISO as they were for the Rainbow Books.

As indicated in table 4, there are several elements in the catalogue of ISO protocols which will only be clarified over the next few years. For today's implementors anxious to become compatible with ISO, some element of guess work is needed as to how the eventual standards might turn out. The UK Department of Industry has established an Information Technology Standards Unit with the objective of formulating an Intercept Strategy. This will include definitions of likely future standards, based on the most authoritative advice.

The UK academic community is paying close attention to the Intercept Strategy. Some of the material in section 9 is based on information in the strategy documents. Planning has already started on a transition path from Rainbow to ISO including moves from Rainbow YBTS to ISO Network Service and from Rainbow JTIP to ISO JTM.

12. CONCLUSIONS

The OSI model provides a useful framework for considering complex communications systems. It is not necessarily the only one or even the best one but it is regarded as adequate for structuring the standardisation process. At present, enthusiastic implementors are in something of a dilemma because, whilst the model is reasonably solid, there is still much to be done before all the relevant protocol standards are ratified. The best that can be done in the gaps is to implement the best guesses at what might

emerge but to maintain sufficient flexibility in the products to allow complete replacement in the years to come.

The UK academic community's experience of a pre-ISO Open Systems project contains important lessons for LEP and for others embarking on similar journeys. Where there is a variety of equipment from different manufacturers to be interconnected, it is highly desirable to implement a solution based on OSI principles in advance of full formal standards. As with any sizeable project, an unambiguous strategy is essential and this must be backed up by central coordination. The task is made infinitely easier if significant funds are made available to the coordinators for disposal as they see fit. The use of large amounts of project manpower on development of protocol implementations or communications components is undesirable mainly because of long-term maintenance complications. It is far better to involve commercial organisations in as many activities as possible. Procurement of equipment is an important means of devolving responsibilities onto manufacturers. Operational requirements should therefore include mandatory clauses on protocols. At an early stage, preparation should begin for the eventual introduction of manufacturers' products which will conform to future ISO standards. Finally, a word of warning is in order. Manufacturers' claims of commitment to ISO and the adherence of their present and future products to ISO standards should be subjected to the closest scrutiny.

REFERENCES

- * 1) Intercept Recommendations for Local Area Networks according to the CSMA/CD Access Method, IT Standards Unit TG101/2 (1984).
- * 2) Cambridge Ring 82 Interface Specifications, IT Standards Unit ITSU/1014 (1982).
- * 3) Intercept Recommendations for the OSI Network Layer, IT Standards Unit TG100/1 (1984).
- 4) OSI - Standard Architecture and Protocols, Proc. IEEE, Vol. 71, No. 12 (Special issue).
- 5) P F Linington, Progress in Open Systems Standardization, Interfaces in Computing Vol. 2, No. 3, pp 205-220 (1984).
- 6) J Larmouth and R A Rosner, "Networking Protocols in the UK Academic Community" in Network Architectures (ed. C. Solomonides) (Pergamon Infotech, 1982), pp 143-158.
- * 7) A Network Independent File Transfer Protocol (Blue Book), IT Standards Unit ITSU/1011 (1981).
- * 8) A Network Independent Job Transfer and Manipulation Protocol (Red Book), IT Standards Unit ITSU/1012 (1981).

General Reading

- 9) A S Tanenbaum, Computer Networks (Prentice-Hall, 1981).
- 10) A S Tanenbaum, Network Protocols, Computing Surveys Vol. 13, No. 4, pp 453-489 (1981).
- 11) R A Rosner, From Copernicus to Computer Networking, Interfaces in Computing Vol. 1, No. 2, pp 95-104 (1983).
- 12) R Cole, Computer Communications (Macmillan, London, 1981).
- 13) W Stallings, Local Networks - An Introduction (Macmillan, New York, 1984).
- 14) K.C.E. Gee, Introduction to Local Area Computer Networks (Macmillan, London, 1983).

* These documents are available from

IT Standards Unit
Department of Trade and Industry
29 Bressenden Place
London, SW1E 5DT

THE NEWCASTLE CONNECTION:

a software subsystem for constructing distributed UNIX*
systems

B. Randell

Computing Laboratory,
University of Newcastle upon Tyne

ABSTRACT

The Newcastle connection is a software subsystem that can be added to each of a set of physically interconnected UNIX or UNIX look-alike systems, so as to construct a distributed system which is functionally indistinguishable at both the user and the program level from a conventional single-processor UNIX system. The techniques used are applicable to a variety and multiplicity of both local and wide area networks, and enable all issues of inter-processor communication, network protocols, etc., to be hidden. A brief account is given of experience with such distributed systems, the first of which was constructed in 1982 using a set of PDP11s running UNIX Version 7, and connected by a Cambridge Ring - since this date the Connection has been used to construct distributed systems based on various other computers and versions of UNIX, both at Newcastle and elsewhere. The final sections compare our scheme to various precursor schemes and discuss its potential relevance to other operating systems.

1. INTRODUCTION

The Newcastle Connection is the name we have given to a software subsystem which enables a distributed system to be constructed out of a set of standard UNIX systems. Such distributed systems (which can use a variety and multiplicity of both local and wide area networks) are functionally indistinguishable, at both 'shell' command language level and at system call level, from a conventional centralised UNIX system [1]. Thus all issues concerning network protocols, and inter-processor communication are completely hidden. Instead all the standard UNIX conventions, e.g. for protecting, naming and accessing files and devices, for inter-process communications, for input/output redirection, etc., are made applicable, without apparent change, to the distributed system as a whole.

The Newcastle Connection can be installed without any modification to any existing source code, of either the UNIX operating system, or any user programs. The technique is therefore not specific to any particular implementation of UNIX, but instead is applicable to any UNIX look-alike system that claims, and achieves, compatibility with the original at the system call level.

In subsequent sections we discuss the structure of such distributed systems, (which for the purposes of this paper we will term UNIX United systems), the internal design of the Newcastle Connection, the networking issues involved, some interesting extensions to the basic scheme, our operational experience with it to date, its relationship to prior work and its potential relevance to other operating systems.

2. UNIX UNITED

A UNIX United system is composed out of a (possibly large) set of inter-linked standard UNIX systems, each with its own storage and peripheral devices, accredited set of users, system administrator, etc. The naming structures (for files, devices, commands and directories) of each component UNIX system are joined together in UNIX United into a single naming structure, in which each UNIX system is to all intents and purposes just a

*UNIX is a Trademark of Bell Laboratories.

directory. Ignoring for the moment questions of accreditation and access control, the result is that each user, on each UNIX system, can read or write any file, use any device, execute any command, or inspect any directory, regardless of which system it belongs to. The simplest possible case of such a structure, incorporating just two UNIX systems, is shown below.

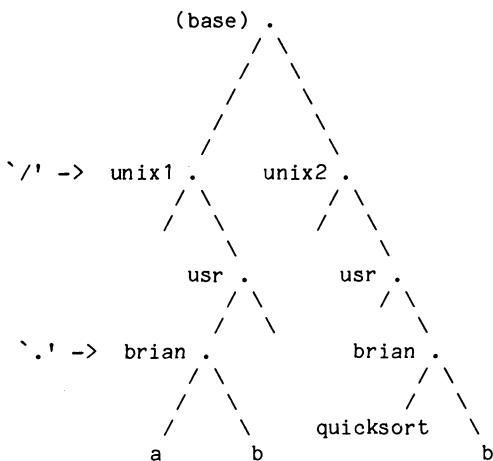


Figure 1: A Simple UNIX United System

With the root directory (`/') positioned as shown, one could copy the file `a' into the corresponding directory on the other machine with the shell command

```
cp /user/brian/a ../../unix2/user/brian/a
```

(For those unfamiliar with UNIX, the initial `/' symbol indicates that a path name starts at the root directory, and the `..' symbol is used to indicate the parent directory.)

Making use of the current working directory (`.') as shown, this command could be abbreviated to

```
cp a ../../unix2/user/brian/a
```

If the user has set up the shell variable `U2' as follows

```
U2=../../unix2/user/brian
```

it could be called forth, using the `'\$' convention, so as to permit the further abbreviation

```
cp a ${U2}/a
```

All the above commands are in fact conventional uses of the standard 'shell' command interpreter, and would have exactly the same effect if the naming structure shown had been set up on a single machine, with `unix1' and `unix2' actually being conventional directories.

All the various standard UNIX facilities (whether invoked via shell commands, or by system calls within user programs) concerned with the naming structure carry over unchanged in form and meaning to UNIX United, causing inter-machine communication to take place as necessary. It is therefore possible, for example, for a user to specify a directory on a remote machine as being his current working directory, to request execution of a program held in a file on a remote machine, to redirect input and/or output, to use files and peripheral devices on a remote machine, etc. Thus, using the same naming structure as before, the further commands

```
cd ../../user/brian  
quicksort a > ../../user/brian/b
```

have the effect of applying the quicksort program on unix2 to the file `a' which had been copied across to it, and of sending the resulting sorted file back to file `b' on unix1. (The command line

```
../../user/brian/quicksort ../../user/brian/a > b
```

would have had the same effect, without changing the current working directory.)

It is worth reiterating that these facilities are completely standard UNIX facilities, and so can be used without conscious concern for the fact that several machines are involved, or any knowledge of what data flows when or between which machines, and of which processor actually executes any particular programs. (Programs are actually executed by the processor in whose file store the program is held, and data is transferred between machines in response to normal UNIX read and write commands.) Moreover all standard UNIX facilities, even the system call used to reposition the root directory, are provided in UNIX United.

In fact what we have done in UNIX United is take advantage of an important but unusual property that UNIX possesses: all file naming is context-relative, in the sense that one can only name files relative to either the current or the root directory, both of which can be re-positioned (but again only using context-relative names). There is thus no way of naming files relative to any absolute point, such as the base of the tree. This feature of UNIX is more commonly used to enable multiple UNIX file systems to be held within one machine, but it is equally useful for splitting up a file system over a number of machines.

2.1. User Accreditation and Access Control

UNIX United allows each constituent UNIX system to have its own named set of users, user groups and user password file, its own system administrator (super-user), etc. Each constituent system has the responsibility for authenticating (by user identifier and password) any user who attempts to log in to that system.

It is possible to unite UNIX systems in which the same user identifier has already been allocated (possibly to different people). Therefore when a request, say for file access, is made from system `A', of system `B', on behalf of user `u', the request arrives at `B' as being from, in effect user `A/u' - a user identifier which would not be confused with a local user identifier `u'. It will be, in effect, this user identifier `A/u' which governs the uses by `u' of files, commands, etc., on machine `B'.

Just as the system administrator for each machine has responsibility for allocating ordinary user identifiers, so he also has responsibility for maintaining a table of recognised remote user identifiers, such as `A/u'. If the system administrator so wishes, rather than refuse all access, he can allow default authentication for unrecognised remote users, who might for example be given `guest' status - i.e. treated as if they had logged in as `guest', presumably a user with very limited access privileges.

From an individual user's point of view therefore, though he might have needed to negotiate not just with one but with several system administrators for usage rights beforehand, access to the whole UNIX United system is via a single conventional log in. Subject to the rights given to him by the various system administrators, he will then be governed by, and able to make normal use of, the standard UNIX file protection control mechanisms in his accessing of the entire distributed file system. In particular there is no need for him to log in, or provide passwords, to any of the remote systems that his commands or programs happen to use. This approach therefore preserves the appearance of a totally unified system, without abrogating the rights and responsibilities of individual system administrators.

At the other extreme, so to speak, it is possible to use the mechanisms we have provided to set up a UNIX United system in which there is, in effect, just a single system administrator, and a single set of accredited users. Then any user can sign on, in the same way, to any of the UNIX systems, and the system administrator can readily control, and perform system maintenance tasks relating to, the entire UNIX United system.

2.2. The Structure Tree

The naming structure of the UNIX United system represents the way in which the component UNIX system are inter-related, as regards naming issues. When a large number of systems are united, it will often be convenient to set up the overall naming tree so as to reflect relevant aspects of the environment in which the UNIX systems exist. For example, a UNIX United system set up within a university might have a naming structure which matches the departmental structure.

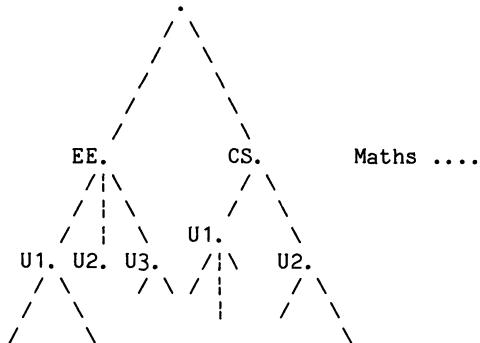


Figure 2: A University-wide System

With the naming structure as shown, files in the system 'U1' in the Computing Science Department could be named using the prefix `../../../../CS/U1' from within the Electrical Engineering Department's UNIX systems.

Such a naming structure has to be one that can be agreed to by all the system administrators, and which does not require frequent major modification - such modification of the UNIX United naming structure can be as disruptive as a major modification of the structure inside a single UNIX system would be, due to the fact that stored path names (e.g. incorporated in files and programs) could be invalidated.

The naming structure could, but does not necessarily, reflect the topology of the underlying communications network. It certainly is not intended to be changed in response to temporary breaks in communication paths, or of service from particular UNIX systems. (An analogy is to the international telephone directory - the UK country code (44) continues to exist whether or not the transatlantic telephone service is operational.)

One final point: We have developed mechanisms which make it possible for UNIX systems to appear in the naming structure in positions subservient to other UNIX systems, though these are not yet incorporated in the version of the Connection which we make available to other organizations. For example, in the previous figure, CS might denote a UNIX system, not just an ordinary directory. We regard this as a very important generalisation, since it allows existing UNIX United systems to be combined together, just as if they were ordinary UNIX systems.

3. THE NEWCASTLE CONNECTION

The UNIX United scheme whose external characteristics were described above is provided by means of communication links, and the incorporation of an additional layer of software - the Newcastle Connection - in each of the component UNIX systems. Conceptually, this layer of software sits on top of the resident UNIX kernel, i.e. between the UNIX kernel and the rest of the UNIX software (e.g. shell and the various command programs) and the user programs. In actual fact, one has a choice between keeping the Connection completely separate from the kernel, or of installing it within the kernel. The former is the simpler means of installing the Connection, but involves the recompilation or relinking of existing user programs and non-resident UNIX software. The latter technique is a kernel-specific optimization that avoids the need for such recompilation or relinking. This method of installation naturally requires more effort and experience, and is best undertaken after completing the simpler porting technique, but in practice has not proved overly difficult. For convenience, in what follows we will assume that the Connection has been installed as a software layer, separate from the kernel.

From above, the Connection layer is functionally indistinguishable from the kernel. From below, it appears to be a normal user process. Its role is to filter out system calls that have to be re-directed to another UNIX system, and to accept system calls that have been directed to it from other systems. Communication between the Connection layers on the various systems is based on the use of a remote procedure call protocol [2], and is shown schematically below:

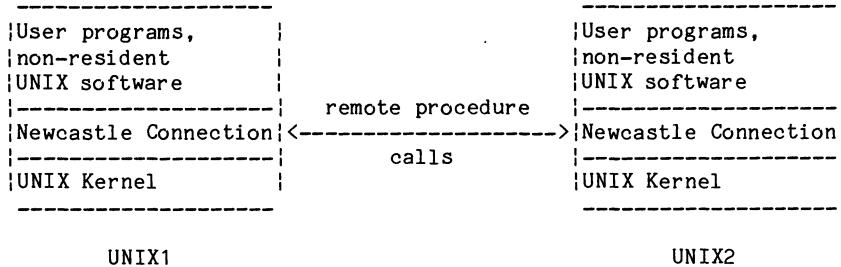


Figure 3: The Position of the Connection Layer

In fact a slightly more detailed picture of the structure of the system would of course reveal that communications actually occur at hardware level, and that the kernel includes means for handling low level communications protocols.

The Connection layer has to disguise from the processes above it the fact that some of their system calls are handled remotely (e.g. those concerned with accessing remote files). It similarly has to disguise from the kernel below it that the requests for the kernel's services, and the responses it provides, can be coming from and going to, remote processes. This has to be done without in any way changing the means by which system calls (apparently direct to the UNIX kernel) identify any real or abstract objects that are involved.

The kernel in fact uses various different means of identification for the various different types of object. For example, open files (and devices) are identified by an integer (usually in the range 0 to 19), logged on users by what is effectively an index into the password file, etc. Such name spaces are of course inherently local. The Connection layer therefore has to accept such an apparently local name and use mapping tables to determine whether the object really is local, or instead belongs to some other system (where it may well be known by some quite different local name). The various mapping tables will have been set up previously - for example when a file is opened - and for non-local objects will indicate how to communicate with the machine on which the object is located. The selection of actual communication paths is performed by the Connection layer, and completely hidden from the user and his programs.

Such mapping does not however apply to the single most visible name space used by UNIX, i.e. the naming structure used at shell level, and at the program level in the 'open' and 'exec' system calls, for identifying files and commands, respectively. Rather, the Connection layer can be viewed as performing the role of glueing together the parts of this naming structure that are stored on different UNIX machines, to form what appears to be a single structure. Each component UNIX system stores, firstly, the section of the naming tree associated with the system's own files and devices. Secondly, each system also stores a copy of those parts of the overall naming structure that relate it to its "name neighbours". These are the other UNIX systems with which it is directly connected in naming terms (i.e. which can be reached via a traversal of the naming tree without passing through a node representing another UNIX system).

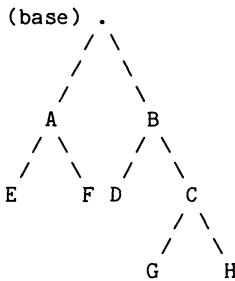


Figure 4(a): A UNIX United Name Space

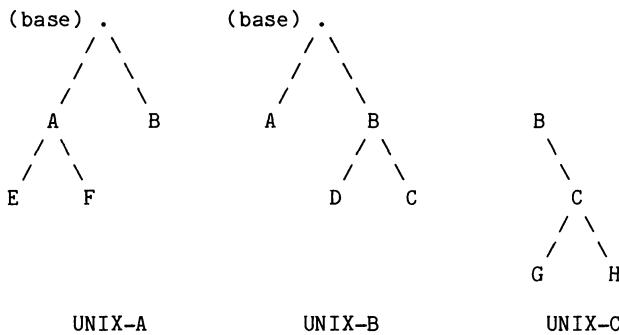


Figure 4(b): Representation of the Name Space

In Figure 4(a), if "directories" A, B and C are associated with separate UNIX systems, the parts of the tree representation stored in each system are as shown in Figure 4(b), namely:

UNIX-A: A,B,E,F,(base)

UNIX-B: A,B,C,D,(base)

UNIX-C: B,C,G,H

It is assumed that shared parts of the naming tree are agreed to by the administrators of each of the systems involved, and do not require frequent modification - a major modification of the UNIX United naming structure can be as disruptive as a major modification of the naming structure inside a single UNIX system. This is because names stored in files or incorporated in programs (or even just known to users) may be invalidated. (Again one can draw a useful analogy to the telephone system. Changes to international and area codes would be highly disruptive, and are avoided as far as possible. For example, they are not changed merely because the underlying physical network has to be modified.)

Within each UNIX system, the Connection uses the local fragment of the UNIX United naming tree to resolve file names. Names are interpreted as a route through the tree, each element specifying the next branch to be taken. If the name can be fully interpreted locally, only a local access is involved. If a leaf corresponding to a remote system is reached, then execution must be continued remotely by making a remote procedure call to the appropriate system. Such leaves are specially marked, and contain the network address of the appropriate remote station. This address is given to the RPC as routing information. (In some cases a request may be passed on through a number of Connections before being satisfied.)

As well as accessing files using a name, a UNIX program can 'open' a file and thereafter access it using the file descriptor returned from the 'open' system call. When a file is opened the Connection makes an entry in a per-process table indicating whether or not the file descriptor refers to a local or a remote file. The table also holds network station addresses for remote file descriptors. Subsequent accesses using the

descriptor refer to this table using the information there to route remote accesses without further delay.

The actual remote file access is carried out for the user by a file server process that runs in the remote machine. Each user has their own file server, and the initial allocation of these is carried out by a "spawner" process that runs continuously. This latter process is callable (using a standard name) by any external user, and, upon request, will spawn a file server (after carrying out some user/group mapping), returning its external name to the user that initiated the request. The user then communicates directly with this file server, which is capable of carrying out the full range of Unix system calls. The user/group mapping is carried out to ensure that the access rights of the file server are in accord with those allowed to the external user by the local system manager, and consists of converting external names into valid local names. Nevertheless, a file server is still an extension of the environment of a user on a remote machine, and any relevant changes in the environment seen by a user must be mirrored by it. The most important of these is that when a user process "forks" (that is, creates a duplicate of itself), all the remote file servers which it is connected with must also fork. This greatly simplifies the implementation of remote execution and signalling, as each user process only ever has to deal with a single remote file server.

Communication with the "spawner" and the file servers always takes the form of a remote procedure call, the first parameter of all calls being a sequence number. This is used by the servers to detect retry attempts - if the received sequence number is the same as that of the last call, then it is a retry (the RPC scheme precludes calls being lost, so there is no need to check for continuity in the sequence).

4. NETWORKING ISSUES

As indicated above, all communication between machines in a UNIX United system is performed by means of the RPC protocol, using network addresses that have been obtained from the leaves of the local fragment of the naming tree. Ideally, all the machines will be directly connected together, i.e. will belong to the same network address space, so that any machine can make a remote procedure call directly on any other machine. This is immediately achieved by the use of a single physical network, such as an Ethernet, but could be also be achieved by some sort of inter-network transport service which hides the existence of multiple physical networks from the Connection. However the Connection is currently being extended to contain its own provisions for coping with multiple network address spaces, and for forwarding RPC calls across networks, for use in situations where such a transport service is not provided.

Any actual implementation of the RPC protocol requires primitive operations for exchanging messages between processes on different machines. In order to shield the Connection layer from the complexities of having to handle differing network interfaces (for reasons both of simplifying its design, and improving its portability) we have recently defined a single interface, the UDS interface, which provides a uniform process-to-process datagram service [3]. This hides the actual protocols used over each local or wide area network, and provides instead a small set of simple primitives for sending and receiving (possibly large) datagrams, using a standardized network addressing scheme, based on a <host number, port number> pair. Exception reporting is also standardized, the assumption being that the implementation of the UDS interface contains, where necessary, sufficient fault tolerance measures for the Connection to be able to rely on a datagram being transmitted accurately, and in its entirety, unless an exception is reported, in which case the Connection can request a retry. (The fault tolerance measures taken by any particular implementation will depend on the assumptions that can be made about the inherent reliability of the actual network involved.)

The UDS interface enables a datagram to be sent from, or received into, a set of non-contiguous buffers, and in effect places no further limit on the size of the datagram than that implied by the total size of the buffers. Each I/O driver implementing the interface for a given network therefore has the responsibility for performing any fragmentation and reassembly operations made necessary by the limitations imposed by the underlying communications hardware and software. This shields the rest of the Connection from such limitations, which in any case tend to be network-specific. The scatter/gather facilities provided to enable the use of non-contiguous buffers greatly reduce the amount of data copying involved in an RPC, and the ability to send large datagrams similarly reduces the number of I/O calls that have to be made across the user/kernel interface. Thus we have found that the introduction of the UDS interface in place of more

conventional network-specific datagram interfaces has not only simplified the tasks of the Connection, and the problems of porting it onto new hardware, but also given useful performance benefits [4].

5. EXTENSIONS TO THE BASIC UNIX UNITED SCHEME

We have found that the conceptual simplifications to the task of implementing a UNIX-based distributed computing system that the Newcastle Connection approach has provided have spurred us to produce a variety of extensions of, or variations on, the basic theme, some of which we have already started to implement.

The Connection layer can be regarded as isolating and solving the problems associated just with distribution - and, it turns out, is applicable to the case of distributed systems made from components other than complete UNIX systems. For example, one could connect together some systems which have little or no file storage with other systems that have a great deal - i.e. construct a UNIX United system out of workstations and file servers. Almost all that is necessary is to set up the naming tree properly.

Moreover since the Connection layer can be independent of the internals of the UNIX kernel, it is not even necessary for the Connection layer to have a complete kernel underneath it - all that is needed is a kernel that can respond properly (even if only with exception messages) to the various sorts of system call that will penetrate down through, or are needed to support, the Connection layer. In fact the Connection layer itself can be economised on, if for example it is mounted on a workstation that serves as little more than a screen editor, say, and so has only a very limited variety of interactions with the rest of the UNIX United system. All that is necessary is adherence to the general format of the inter-machine system call protocol used by the Newcastle Connection, even if most types of call are responded to only by exception reports.

Thus the syntax and semantics of this protocol assume a considerable significance, since it can be used as the unifying factor in a very general yet extremely simple scheme for putting together sophisticated distributed systems out of a variety of size and type of component - an analogy we like to make is that the protocol operates like the scheme of standard-size dimples that allow a variety of shapes of LEGO children's building blocks to be connected together into a coherent whole.

In addition to the problem of distribution, we also have taken what are, we believe, several other equally separable problems, in particular those of (i) providing error recovery (for example in response to input errors or unmaskable hardware faults), (ii) using redundant hardware provided in the hope of masking hardware faults, (iii) the enforcement of multi-level security policies and (iv) load balancing between the component systems, and plan wherever practicable to embody their solutions in other separate layers of software. Indeed, three significant extensions of UNIX United have already been implemented, albeit in prototype form. The first of these provides multi-level security, using encryption to enforce security barriers between component machines (which each run at a single security level) and to control permissible security re-classifications [5]. The other two extensions are related to hardware fault tolerance. One uses file and process triplication and majority voting to mask hardware faults - application programs are unchanged, though in fact running in synchronisation on several machines with hidden voting. The other uses duplicated disks to provide a crash-resistant high integrity file system [6].

6. OPERATIONAL EXPERIENCE

The first UNIX United system was based on a set of three PDP11/23s and two aged PDP11/45s, all running UNIX V7, and connected by a Cambridge Ring. At the time of writing (August 1984) this system is being upgraded, with the 11/45s being replaced by VAX/750 computers. The system has been operational for over two years, and usually in regular daily use. Our experience has been that the most heavily used facilities have been those concerned with file transfer and I/O redirection, for example in order to make use of the line printer and magnetic tape unit that are attached to one machine. The Connection has also been relied on for network mail, for solving the problems of overnight file-dumping (of all machines, onto the one tape unit) and, perhaps most significantly, for software maintenance and distribution within the UNIX United system.

A second and hitherto separate UNIX United system at Newcastle, based on ICL PERQs connected by Ethernet, was implemented during May 1983 in collaboration with ICL, and has

since also been used regularly. Work is now in hand to link this system to the VAX/750 computers, and to several other recently acquired UNIX machines, from various manufacturers, in order to produce one single enlarged and somewhat heterogeneous UNIX United system, involving both a Ring and an Ethernet.

Pre-release versions of our software were first made available to several other organisations, starting in mid-1982, the first formal release being issued in June 1983. By now a considerable number of organizations have taken out either commercial or educational licenses, and have ported the Connection to various other machines, networks and versions of UNIX, including System III and Berkeley 4.2.

As regards the performance of a UNIX United system, it is clear that this depends on three essentially separate factors: the capabilities of the component UNIX systems, the efficiency of the underlying communications hardware and software, and the overheads due to the Connection, only the last of which is our responsibility. In fact the overheads due to the Connection are really quite modest. Those caused by the need to confirm that a system call only involves a local file descriptor are virtually imperceptible, though local path name calls such as 'open' and 'exec' are slowed down somewhat, since for each such call an additional 'stat' system call is made from within the Connection. When a call proves to involve a remote facility, this normally just involves making one RPC call, and waiting for a reply. The RPC protocol is itself very simple and usually involves sending one message (the packaged system call) and receiving one message in reply. The file server that accepts such calls from remote machines is similarly simple, being dedicated to serving the needs of just a single remote process, and in most cases does little more than make system calls on behalf of this process and send it the results.

Our first UNIX United system functioned surprisingly well, despite the fact that the Cambridge Ring stations used were quite slow, being interrupt-driven rather than direct memory access devices. (Such stations cause UNIX to take an interrupt for every pair of bytes sent and received over the Ring!) In fact terminal users in general noticed little performance difference between local and remote accesses and execution. This perhaps indicates that even interrupt-driven stations are reasonably well matched to the rather modest performance that UNIX itself can achieve on a small PDP11/23 used as a personal workstation, or on a PDP11/45 that is usually being used by a number of demanding terminal jobs.

A separate project has now been set up to undertake performance monitoring and evaluation of UNIX United systems, a task whose difficulty derives in part from the well-known problems of making meaningful performance assessments and comparisons of ordinary UNIX systems. However some simple experimental measurements have already been made using our PERQ-based system, which has much more adequate network hardware, in fact an Ethernet with direct memory access interface units. These measurements produced the initially surprising result that copying of files to or from a remote PERQ could be 20% or more faster than local file copying. In fact this merely indicates the extent to which contention for a single disk can limit a machine's performance. One other interesting measurement showed that file transfers using the standard UNIX file copy command and the Connection achieved almost twice the speed achieved by the manufacturer-supplied file transfer protocol, which uses the ECMA Level 4 Transport Service over the Ethernet. However the more significant result is that, on this system also, users in general notice little difference in performance between local and remote operations.

The total amount of code involved in the various parts of the Connection is about 11,000 lines of C. Of this, the file server code amounts to approximately 2500 lines, the code involved in intercepting and mapping the various system calls some 7500 lines, and the 'spawner' which is used to start up file server processes on demand the remaining 1000 lines. Installation of the Connection as a separate layer involves including a copy of selected parts of the interception code in each user program. On the PDP 11/45, for example, the amount of code added varies between 3.5k and 12k bytes, depending on the number of different system calls that the program invokes. (The alternative means of installing the Connection, discussed briefly in Section 3, involves inserting just the interception code in the kernel - the file server and spawner code remains outside the kernel.)

On the PDP 11/45 the file server code occupies about 12.5k bytes, and in addition each actual server process requires 2.5k bytes of space. The single spawner process requires a total of 8k bytes of code and data space. (By way of comparison, the UNIX kernel as set up for our particular I/O configuration occupies 48.5k bytes of code space, and 83.5k bytes in total.) The comparatively small size of the Connection reflects the need we

had to make the system work on our small PDP11/23s, which provided a strong incentive to find what we feel justified in claiming are simple well-structured solutions to the various implementation problems. (In our view an overabundance of program storage space can have almost as bad an effect on the quality of a software system as does inadequate space - it is surely no coincidence that UNIX was first designed for quite modestly sized machines!)

7. RELATED EARLIER WORK

The Newcastle Connection, and the UNIX United scheme that it makes possible, have many precursors, and not just within the UNIX world.

The idea of providing a layer of software which aims to shield users of a set of inter-connected computers from the need to concern themselves with networking protocols, or even the fact of there being several computers involved, is well-established. It is, for example, what the IBM CICS System [7] does for users of various transaction-processing programs, and what the National Software Works project [8] aimed to do for the users of various software development tools, running on a variety of different operating systems. Such layers of software are intended for somewhat specialised use, and run on top of specific sets of application programs. At the other end of the spectrum, such location- or network-transparency is also one of the aims of the Accent kernel [9], on which operating systems can be constructed which use its "port" concept as a means of unifying inter-process communication, inter-computer message passing, and operating system calls.

The dawning realisation that the 'shell' job control language and the program-level facilities (i.e. system calls) of the UNIX multiprogramming system could suffice, and indeed would be highly appropriate, to control a distributed computing system can be traced in a whole series of distributed UNIX projects. The global file naming technique used in the early 'uucp' facilities [10] for interconnecting UNIX systems via standard telephone circuits can be seen as a special, but rather ad hoc, extension of the individual file system naming hierarchies, and had been copied by us in our Distributed Recoverable File System [11]. (The technique provides what is in effect a set of named hierarchies, rather than a single enlarged hierarchy.)

Rather better integrated with the standard UNIX file naming hierarchy are the facilities provided in the Network UNIX System [12]. This modification of standard UNIX provides a series of Arpanet protocols, which are invoked by means of some additional system commands, using what appear to be ordinary file names as the means of identifying which Arpanet host is to be communicated with. (The paper describing this system also speculates on the possibility of redesigning the shell interpreter so as to provide network transparency for commands and files at the shell command language level.) The Purdue Engineering Computer Network [13] is conceptually similar to the Network UNIX System, though based on hard-wired high speed duplex connections. It provides additional commands which invoke the services of special protocols for virtual terminal access and remote execution at the shell level, and also a means of load balancing through a scheduling program which takes responsibility for deciding which processor should execute certain selected commands.

The distributed system of interconnected S-UNIX personal workstations and F-UNIX file servers [14] goes further by providing each workstation user with a ordinary UNIX interface, without any additional non-standard commands, yet incorporating a distributed version of the UNIX hierarchical file store containing just his own local files plus all the files held on the file servers. This system is one of several built at Bell Labs using the Datakit virtual circuit switch - others are RIDE [15] and D/UNIX [16]. The RIDE system provides complete remote file access and remote program execution, but is based on a 'uucp'-like, rather than standard, UNIX naming hierarchy - it is however implemented merely by adding a software layer on top of the UNIX kernel, an approach which is highly similar to that we have since used with our Newcastle Connection technique. D/UNIX is a distributed system based on modified versions of UNIX which provide virtual circuits between processes, and a transparent file sharing scheme covering all the files on all the component systems.

A fully symmetrical means of linking computer systems together so as to give the appearance of a single UNIX-like hierarchical file store, and the standard shell command language, is also provided by the LOCUS system - the paper [17] describing this system also discusses its intended extension to provide remote program execution as well as remote file access. However, for all its external similarity to UNIX, the LOCUS system

involves a completely redesigned operating system rather than a modification of an existing UNIX system, albeit an operating system which is also designed to have extensive fault tolerance facilities.

The penultimate stage in the evolution can be seen in the COCANET local network operating system [18], a system which has been built using the standard UNIX system, and which comes very close indeed to our aim of combining a set of standard UNIX systems into a single unified system, and which certainly supports network-transparent remote execution as well as file access. However the COCANET designers have allowed themselves to make a number of changes to the UNIX kernel and would appear, from the description they give, not to have coped fully with user-id mapping. It would also appear that COCANET is designed specifically around the idea of having a relatively small number of machines linked by a single high-speed ring, and hence has a rather restrictive structure tree, which is viewed slightly differently from each machine. However many of the mechanisms incorporated in UNIX United are very similar to those used in COCANET.

It is thus but a comparatively small step from COCANET to UNIX United, and to the idea of the Connection layer capable of being placed on top of an unchanged UNIX kernel, replicating all its facilities exactly in a network-transparent fashion, and capable of making a distributed system involving large numbers of computers, connected by a variety of local and wide area networks. Incidentally, one can draw an interesting parallel between the Connection layer and what is sometimes called a "hypervisor", the best-known example of which is VM/370 [19]. Each is a self-contained layer of software, which makes no changes to the functional appearance of the system beneath it (the IBM/370 architecture in the case of VM/370, which fits under rather than on top of the operating system kernel). However whereas a hypervisor's function is to make a single system act as a set of separate systems, the Newcastle Connection (a "hypovisor"?) makes a set of separate (though of course linked) systems act like a single system!

However, to our embarrassment, we have to admit that the idea of the Connection layer, of the basic UNIX United scheme, and of most of the extensions of the scheme, did not arise from careful study and analysis of these precursors. (Indeed it is clear that what was presented above as a more-or-less orderly evolutionary development path often involved parallel activity by several groups, and much accidental re-invention.) In fact we were not consciously aware of any of these systems (other than 'uucp' and of course DRFS) whilst the work that led to the Newcastle Connection was in progress. Indeed, by the time we learnt of LOCUS and COCANET, all the basic ideas and strategies to be incorporated in the Newcastle Connection had been worked out, though not all in full detail, and much of the system was already operational and in daily use. Rather we can trace the origins of our scheme to the existence of the plans for our remote procedure call protocol, and the idea, which we now know has occurred to many groups independently, of extending the UNIX 'mount' facility from that of mounting replaceable disk packs to that of mounting one UNIX system on another.

This idea arose at Newcastle in early December 1981 - within a week or so much of the UNIX United concept had been thought up and even roughly documented. A hesitant start at what was initially intended as just an experimental and partial implementation was made after Christmas, but within a month many facilities related to accessing and operating on files remotely over the Cambridge Ring were in active use. Work proceeded rapidly, both on extending the range of UNIX kernel features that the Newcastle Connection mapped correctly, and on discovering, mainly via experimentation, some of the more arcane features of the kernel interface as implemented and used in V7 UNIX. At about this stage we found out about first the S-UNIX/F-UNIX and LOCUS systems, and shortly afterwards the COCANET and then the RIDE systems. These various papers were a considerable encouragement to us to continue our efforts, leading to a first complete system by mid-1982, and also provided us with a useful perspective on our approach. In particular they strengthened our growing belief in the viability of an alternative, UNIX-based, approach to distributed computing to that based on the use of a variety of explicit servers, each with its own specialised service protocol [20,21].

8. WHY JUST UNIX?

It is interesting to analyse just what it is about UNIX, and the linguistic interfaces it provides at shell and system call level, that make it so suitable for use as the model and basis for a network operating system. There seem to be six principal factors involved.

First, there is the hierarchical file (and device and command) naming system. This makes it easy to combine systems, because the various hierarchical name spaces just become component name spaces in a larger hierarchy, without any problems due to name clashes. The standard UNIX mechanisms for file protection and controlled sharing of files then carry over directly, once the problem of possible clashes of user identifiers is handled properly.

Second, there are the UNIX facilities for dynamically selecting the current working directory and root directory. In particular the ability to select the root directory - normally thought of as one of the more exotic and little needed of the UNIX system commands - seems to have been designed especially for UNIX United, since it provides a perfect way of hiding the extra levels of the directory tree that have to be introduced.

Third, and obviously vital, is the fact that UNIX allows its users, and their programs, to initiate asynchronous processes. This is used inside the Newcastle Connection, and also provides the means whereby even a single user can make use via the Newcastle Connection of several or indeed all of the computers that are involved in the UNIX United system. It also provides the means whereby slow file transfers (via low bandwidth wide area networks) can be relegated to background processing, and so still be organised using remote procedure calls.

Fourth, there is the fact that the UNIX system call interface is (relatively) clean and simple, and can easily be regarded as providing a small number of reasonably well defined abstract types. The task of virtualising these types, so as to give network transparency, therefore remains manageable.

Fifth, there is the fact, even in this day and age still regrettably worthy of mention, that the original UNIX system, and all of its derivatives known to us, are written in a fairly satisfactory high level language. Our method of incorporating the Newcastle Connection into UNIX as a separate software layer therefore merely involved recompiling relevant parts of the system, using a different subroutine library.

Finally, there is the well-established set of exception reporting conventions that are used in UNIX, for example, to indicate the reasons why particular system call requests cannot be honoured. When such a call has, via the Newcastle Connection, involved attempted communication with another UNIX system there are various other (quite likely) reasons, but they can be mapped onto the exceptions that the caller is already supposed to be able to deal with.

However it is unlikely that the idea could not be carried across to at least some other systems. Indeed a report by Goldstein et al [22] implies that something similar is being bravely contemplated for IBM's MVS operating system, and some aspects of the idea are we understand commercially available as additions to the RSX/11 operating system - no doubt other examples exist. The one other system whose suitability for the Newcastle Connection approach has been considered at all seriously by us is DEC's VAX/VMS system. It would appear that it has many of the necessary characteristics though there could be problems with the way that devices are involved with its system of file naming.

This section would not be complete without any mention of what we regard as some shortcomings of the UNIX V7 specifications (at system call level): Firstly, the system of 'signals' for asynchronous communication between processes could be improved. Allied to this, a general synchronous inter-process communication mechanism would be useful, allowing communication between numbers of unrelated processes. Some awkward features in the file protection scheme were encountered when constructing the file server. These were associated with the notions of 'super-user' and 'effective user-id'. Lastly, we found that the ability to have many directory entries ('links'), each naming the same physical file, was elegant in concept but severely limited in generality by the actual UNIX V7 implementation. (Unfortunately, the various "improved" versions of UNIX that have appeared since Version 7 have done little to remedy most of these shortcomings.)

With respect to the programs that are provided with the UNIX system, very few difficulties were encountered in connecting them, except, that is, for the Shell. This program makes use of system facilities in non-standard ways, and its internal design is obscure to say the least. However, it has proved to be an excellent testbed for the system - when porting the Connection, once the Shell works you can be pretty sure that most other programs will!

9. CONCLUSIONS

The first of our internal memoranda on what we later came to call the Newcastle Connection described the idea as "so simple and obvious that it surely cannot be novel". And, as described above, it did turn out to have a number of precursors - in fact probably many more than we yet realise. However we take this as confirmation of the merits of the twin ideas of network transparency and of its provision by a single separate mapping layer, an approach whose ramifications we feel we have barely begun to explore. (A more general discussion of this approach, and its relevance to the problems of designing highly reliable and secure systems can be found in[23].) Certainly our present plan is to continue our programme of experimental implementations and applications, and to determine how well the Newcastle Connection (and UNIX) can withstand the weight of additional software layers containing the various reliability and security-related mechanisms that we have developed, hitherto in a rather fragmented fashion for various systems and languages.

One other point is worth stressing. It has for some years been well-accepted that the structure and mechanisms of a multiprocessing operating system are very similar to those of a (good) multiprogramming system. What has now become clear to us, as a result of our work on UNIX United, is that this similarity can usefully extend also to distributed systems. The additional problems and opportunities that face the designer of a homogeneous distributed system should not be allowed to obscure the continued relevance of much established practice regarding the design of multiprogramming systems.

10. ACKNOWLEDGEMENTS

These lecture notes are based closely on the original paper describing the Newcastle Connection [24]. My co-authors on this paper, Dave Brownbridge and Lindsay Marshall, jointly implemented the pre-release version of the Connection, since when Lindsay Marshall has borne the major responsibility for its design and implementation, aided recently by Jay Black. The Remote Procedure Call Protocol on which that now used in the Connection is based was originally designed and implemented by Fabio Panzieri and Santosh Shrivastava. The UDS interface was designed by Fabio Panzieri and myself, and has been implemented for various types of machine and network by Fabio Panzieri, Andy Linton, Robert Stroud and Graeme Dixon. Robert Stroud has also had the major responsibility for the first port of the Connection (to the PERQ) and for its integration into a UNIX kernel.

A special acknowledgement is obviously due to the creators of that famous Registered Footnote of Bell Laboratories, UNIX, much of whose external characteristics, if not detailed internal design, deserve the highest praise. Last but not least, I am pleased to acknowledge that our work has been supported by research contracts from the UK Science and Engineering Research Council, and the Royal Radar and Signals Establishment.

* * *

References

- [1] D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System", Comm. ACM Vol. 17(7), pp.365-375 (1974).
- [2] S. K. Shrivastava and F. Panzieri, "The Design of a Reliable Remote Procedure Call Mechanism", IEEE Trans. on Computers (July 1982).
- [3] F. Panzieri and B. Randell, "Interfacing UNIX to Data Communication Networks", Report 190, Computing Laboratory, University of Newcastle upon Tyne (Dec. 1983).
- [4] A. Linton and F. Panzieri, "A Communication System Supporting Large Datagrams on a Local Area Network", Report 191, Computing Laboratory, University of Newcastle upon Tyne (1984).
- [5] J. M. Rushby and B. Randell, "A Distributed Secure System", Computer Vol. 16(7), IEEE (July 1983).
- [6] J. A. Anyanwu, "A Reliable Stable Storage System for UNIX", Report 191, Computing Laboratory, University of Newcastle upon Tyne (1984).

- [7] J. Gray, "IBM's Customer Information Control System (CICS)", Operating System Review Vol. 15(3), pp.11-12, ACM (July 1981).
- [8] R. E. Millstein, "The National Software Works: A Distributed Processing System", Proc. ACM 1977 Annual Conference, Seattle, Washington, pp.44-52, ACM (Oct. 1977).
- [9] R. Rashid, "Accent: A Communication Oriented Network Operating System Kernel", Operating Systems Review Vol. 15(5), pp.64-75 (Dec. 1981).
- [10] D. A. Nowitz, "Uucp Implementation Description", p. Sect. 37 in UNIX Programmer's Manual, Seventh Edition, Vol. 2 (Jan. 1979).
- [11] M. Jegado, "Recoverability Aspects of a Distributed File System", Software Practice and Experience Vol. 13(1), pp.33-44 (Jan. 1983).
- [12] G. L. Chesson, "The Network UNIX System", Operating Systems Review Vol. 9(5), pp.60-66 (1975). Also in Proc. 5th Symp. on Operating Systems Principles.
- [13] K. Hwang, W. J. Croft, G. H. Goble, B. W. Wah, F. A. Briggs, W. R. Simmons, and C. L. Coates, "A UNIX-Based Local Computer Network with Load Balancing", Computer, pp.55-66 (Apr. 1982).
- [14] G. W. R. Luderer, H. Che, J. P. Haggerty, P. A. Kirslis, and W. T. Marshall, "A Distributed Unix System Based on a Virtual Circuit Switch", Proc. 8th Symp. Operating System Principles, Pacific Grove, Calif., pp.160-168, ACM (Dec 1981). Also in: ACM Special Interest Group on Operating Systems - Operating Systems Review, Vol 15, No 5 (Dec 1981).
- [15] P. M. Lu, "A System for Resource Sharing in a Distributed Environment - RIDE", Proc. IEEE Computer Society 3rd COMPSAC, IEEE New York (1979).
- [16] J. C. Kaufeld and D. L. Russell, "Distributed UNIX System", in Workshop on Fundamental Issues in Distributed Computing, ACM SIGOPS and SIGPLAN (15-17 Dec. 1980).
- [17] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin, and G. Thiel, "LOCUS: A Network Transparent, High Reliability Distributed System", Operating Systems Review Vol. 15(5), pp.169-177, ACM (Dec. 1981). (Proc. ACM 8th Conf. Operating System Principles, Asilomar, Calif.).
- [18] L. A. Rowe and K. P. Birman, "A Local Network Based on the UNIX Operating System.", IEEE Trans. on Software Eng. Vol. SE-8(2), pp.137-146 (Mar 1982).
- [19] L. H. Seawright et al, "Papers on Virtual Machine Facility/370", IBM Systems J. Vol. 18(1), pp.4-180 (1979).
- [20] M. V. Wilkes and R. M. Needham, "The Cambridge Model Distributed System", Operating System Review Vol. 14(1), pp.21-28, ACM (Jan. 1980).
- [21] E. Lazowska, H. Levy, G. Almes, M. Fischer, R. Fowler, and S. Vestal, "The Architecture of the EDEN System", Operating Systems Review Vol. 15(5), pp.148-159, ACM (Dec. 1981). (Proc. ACM 8th Conf. Operating System Principles, Asilomar, Calif.).
- [22] B. Goldstein, G. Trivett, and I. Wladawsky-Berger, "Distributed Computing in the Large Systems Environment", Report RC 9027, IBM T. J. Watson Research Center, Yorktown Heights, N.Y. (9 Sept. 1981).
- [23] B. Randell, "Recursively Structured Distributed Computing Systems", pp. 3-11 in Proc. 3rd Symp. Reliability on Distributed Software and Database Systems, IEEE (October 1983).
- [24] D. R. Brownbridge, L. F. Marshall, and B. Randell, "The Newcastle Connection - or UNIXes of the World Unite!", Software Practice and Experience Vol. 12(12), pp.1147-1162 (Dec. 1982).

Unfolding methods in high-energy physics experiments

V. BLOBEL

II.Institut für Experimentalphysik der Universität Hamburg

ABSTRACT. Distributions measured in high-energy physics experiments are often distorted or transformed by limited acceptance and finite resolution of the detectors. The unfolding of measured distributions is an important, but due to inherent instabilities a very difficult problem. Methods for unfolding, applicable for the analysis of high-energy physics experiments, and their properties are discussed. An introduction is given to the method of regularization.

Nay, answer me: stand, and unfold yourself
- William Shakespeare, from Hamlet

You can get it wrong and still you think it's all right
- John Lennon and Paul McCartney, from
We can work it out

1. INTRODUCTION

One of the objectives of experimental physics is to measure distributions $f(x)$ of some physical variable x . Examples from high-energy physics are angular distributions, invariant-mass spectra and structure functions. The set $\{x\}$ of values measured in particle reactions, or events, can be regarded as a random sample, drawn from a distribution $f(x)$ of a one-dimensional random variable x , and it is the purpose of data analysis, to make inferences about $f(x)$ from the set $\{x\}$. The measured distribution $\hat{f}(x)$ ¹, usually determined in the form of a histogram [1], differs from the true distribution $f(x)$ by statistical errors $\epsilon(x)$; it can be used to test theoretical predictions $f_{th}(x)$. In the case of an expression $f_{th}(x, a)$, depending on parameters a , values for the parameters can be extracted by fitting $f_{th}(x, a)$ to the data $\hat{f}(x)$.

In high-energy physics experiments often the quantity x and the distribution of x cannot be measured directly, due to the imperfection of the detector. Two detector effects can be distinguished, limited acceptance and resolution. Limited acceptance means that the probability to observe a given event is less than one; the acceptance may depend on the kinematical region. The second effect, limited resolution, means that the quantity x in a given event cannot be determined exactly, but it can only be measured with a certain measurement error. Both effects result in a distortion of the measured distribution which can be expressed in the following way: instead of the physically relevant variable x , a variable y and its distribution $g(y)$ is measured. Due to a transformation property of the measurement process, the measured variable y may differ from x completely; for a one-dimensional variable x the variable y may be

¹Measured quantities, and quantities, derived from measured quantities, are denoted by a hat, for example $\hat{f}(x)$.

multidimensional. The transformation may also include additional kinematical effects, for example in scattering experiments there may be effects from radiation and Fermi motion (in case of heavy targets) [2].

In this paper only the case is considered, where the distributions $f(x)$, defined over the range $a \leq x \leq b$, and $g(y)$ are related by the convolution integral

$$g(y) = \int_a^b A(y, x) f(x) dx . \quad (1.01)$$

In the theory of integral equations equation (1.01) is called the Fredholm integral equation of first kind. Integral equations occur in many branches of computational physics. Examples from different fields of science are given in [3] and in [4], which also contains many references. The function $A(y, x)$, called a kernel in the theory of integral equations, describes the response of the detector including the transformation from x to y . For a given $x = x_0$, the response of the detector in the variable y is $A(y, x_0)$. In practice the distribution $\hat{g}(y)$ actually measured differs from the expected distribution $g(y)$ by statistical errors $\epsilon(y)$:

$$\hat{g}(y) = \int_a^b A(y, x) f(x) dx + \epsilon(y) \quad (1.02)$$

An accurate determination of the response function $A(y, x)$ is essential for any meaningful analysis of the data. Information on the behaviour of the detector can be obtained by test measurements with a known $f(x)$. For example a hadron calorimeter can be exposed to a particle beam with well known fixed energy $x = x_0$, then $f(x) = \delta(x - x_0)$ and the result of the measurement of the energy y in the calorimeter is directly $A(y, x_0)$:

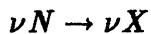
$$\int_a^b A(y, x) \delta(x - x_0) dx = A(y, x_0) . \quad (1.03)$$

Complex high-energy experiments usually require Monte-Carlo calculations of the detector response. This method allows the determination, for a given dependence $f(x)$, of the expected distribution $g(y)$ of any measurable quantity y by a detailed simulation of the measuring process in the detector. Often detector data are generated in the same format as real data, and can be processed by an identical chain of reconstruction and analysis programs.

If the effect of limited resolution is negligible and the measured distributions are essentially affected only by limited acceptance, the correction for these effects is usually not very difficult. There are two possibilities for the data analysis in case of a large distorting effect due to limited resolution. If a specific theoretical prediction $f_{th}(x)$ is to be compared with the data, the corresponding expected distribution $g_{th}(y)$ can be evaluated by equation (1.01). If the distribution $g_{th}(y)$ agrees with the measured distribution $\hat{g}(y)$ within statistical errors, one can conclude, that the tested prediction gives a consistent description of the measured process. In the case of disagreement however a publication of a detector-dependent measured distribution $\hat{g}(y)$ is of little use. The comparison with alternative theoretical predictions is impossible, unless they are also convoluted with the detector response using equation (1.01). The other possibility is the reconstruction of $f(x)$ from the measured distribution $\hat{g}(y)$.

The reconstruction of $f(x)$ from the measured distribution $\hat{g}(y)$ is called unfolding, and it is a statistical estimation problem. Unfolding is a complicated problem, because, in mathematical classification, it is an ill-posed problem [3], which can have wildly oscillating solutions [4]. In spite of its practical importance, it is not mentioned in standard textbooks on statistics, with only few exceptions [5]; in general many misconceptions exist and are used in practical applications. It is characteristic of these methods, that they are usually described only in words; the standard criteria for statistical estimation methods, the requirements for efficiency, consistency and unbiasedness are usually difficult to discuss for these methods.

One example of the combined effects of acceptance, transformation and resolution in a high-energy physics experiment is the measurement of the cross section $d\sigma/d\eta$ as a function of the inelasticity $\eta = (E_h - m_p)/E_\nu$ in neutral current reactions

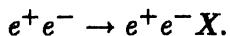


in a narrow band neutrino beam¹ [6]. The detector allows the measurement of the energy E_h of the hadronic system X above a threshold of a few GeV with a certain resolution, and in addition the distance r of the interaction from the beam axis can be measured (m_p is the proton mass). The beamflux $\Phi(E_\nu, r)$ has a two-peak structure at a fixed distance r , since the neutrinos have their origin in decays of π - and K -Mesons. Clearly the measured data do not allow the reconstruction of the value η of the inelasticity in individual events, since the neutrino energy E_ν is not known. The knowledge of the beamflux $\Phi(E_\nu, r)$ together with the known acceptance and resolution in E_h and r allows the calculation for any given $d\sigma/d\eta$ of the resulting distribution of the events in the (E_h, r) -plane. The transformation between η and (E_h, r) is completely defined and can be expressed by the formula

$$g(E_h, r) = \int A(E_h, r, \eta) \frac{d\sigma}{d\eta} d\eta, \quad (1.04)$$

which has the same structure as equation (1.01). Unfolding in this case means the reconstruction of $d\sigma/d\eta$ from the measured distribution $g(E_h, r)$.

Another example from high energy physics is the measurement of the total cross section for $\gamma\gamma$ -interactions, which is possible with e^+e^- storage rings at high energies by a measurement of the reaction [7] [8]



Because some fraction of particles of the hadronic final state X are emitted at small angles with respect to the beams, they escape detection by the detector and the measured invariant mass of the hadronic system is on average smaller than the true invariant mass. A model of the physical process is necessary in this case for the calculation of the response function $A(y, x)$. Model parameters have to be determined from the data.

¹The usual symbol y for the inelasticity is replaced here by the symbol η , to avoid confusion with measured variables, denoted by the symbol y in this paper.

If the variables x and y are discrete variables, the integral in equation (1.01) has to be replaced by a sum. A discrete set can always be mapped on a set of consecutive integers, and therefore one can assume, that the random variables x and y have ranges $1 \dots m$ and $1 \dots n$, respectively. Instead of functions $f(x)$ and $g(y)$ of continuous variables, there is a finite number of elements f_j , $j = 1 \dots m$ and g_i , $i = 1 \dots n$, and the convolution equation can be written in the form

$$\hat{g}_i = \sum_{j=1}^m A_{ij} f_j + \epsilon_i \quad \text{or short} \quad \hat{g} = Af + \epsilon, \quad (1.05)$$

where f is a n -vector, \hat{g} and ϵ are m -vectors, and A is a $m \times n$ matrix. The equation can be interpreted as a discrete approximation of the integral equation (1.02). Any numerical solution of the integral equation for continuous variables will require an approximation by a finite number of elements. The simplest discretization is the representation of the distributions by histograms f and \hat{g} (a more general discretization method is discussed in chapter 4.2).

As mentioned before, the correction is not very difficult, if the dominant effect of the detector is limited acceptance. In this case all elements A_{ij} are zero for $i \neq j$ (with $n = m$) and only the elements A_{jj} , the acceptance probability for bin j , have to be considered. A common method for pure acceptance correction is the following: Monte Carlo events with x -values generated according to some fixed assumption \bar{f} are processed, simulating the detector response, and the accepted events are used to fill a histogram \bar{g} . The binwise ratio \bar{g}_j / \bar{f}_j of the histograms gives the values A_{jj} of the acceptance probabilities for bins j . The correction of the measured bin contents \hat{g}_j is then made according to

$$\hat{f}_j = \hat{g}_j \left(\frac{\bar{f}_j}{\bar{g}_j} \right), \quad (1.06)$$

to obtain corrected bin contents \hat{f}_j . Since $A_{ij} = 0$ for $i \neq j$, the correction factor does not depend on the used MC-input histogram \bar{f} . There are no difficulties or problems with this method, which may be called the factor method, except perhaps if the acceptance probability changes rapidly within a few bins [9].

The situation is completely different, if a correction for limited resolution becomes necessary. The principal difficulty of unfolding is easily demonstrated. For the discrete case with $n = m$ (square matrix A), the straightforward application of standard analysis methods suggests the solution

$$\hat{f} = A^{-1} \hat{g}, \quad (1.07)$$

where A^{-1} is the matrix inverse to A ; this method may be called inversion method. The expectation value $E(\hat{f})$ of the estimate \hat{f} is equal to the true f , provided the measured \hat{g} has no bias, i.e. $E(\hat{g}) = Af$:

$$E(\hat{f}) = E(A^{-1} \hat{g}) = A^{-1} E(\hat{g}) = A^{-1} Af = f. \quad (1.08)$$

An estimate with this desirable property is called consistent. Error propagation yields

$$V(\hat{f}) = A^{-1} V(\hat{g}) A^{-1} \quad (1.09)$$

for the covariance matrix $V(\hat{f})$, calculated from the covariance matrix $V(\hat{g})$ of the measured data. This method is probably tried first by many people, when confronted with an unfolding problem. The result however is often very disappointing, as shown in the numerical example below, and it is understandable that people after trying this method turn to a heuristic method which provides 'better' results.

Example: Unfolding of a distribution of a discrete variable. The case $n = m = 20$ is assumed, with the following response matrix:

$$A = \begin{pmatrix} 0.75 & 0.25 & 0 & & \\ 0.25 & 0.50 & 0.25 & 0 & \\ 0 & 0.25 & 0.50 & 0.25 & 0 \\ & 0 & 0.25 & 0.50 & 0.25 \\ & & 0 & 0.25 & 0.50 \\ & & & & \ddots \end{pmatrix}$$

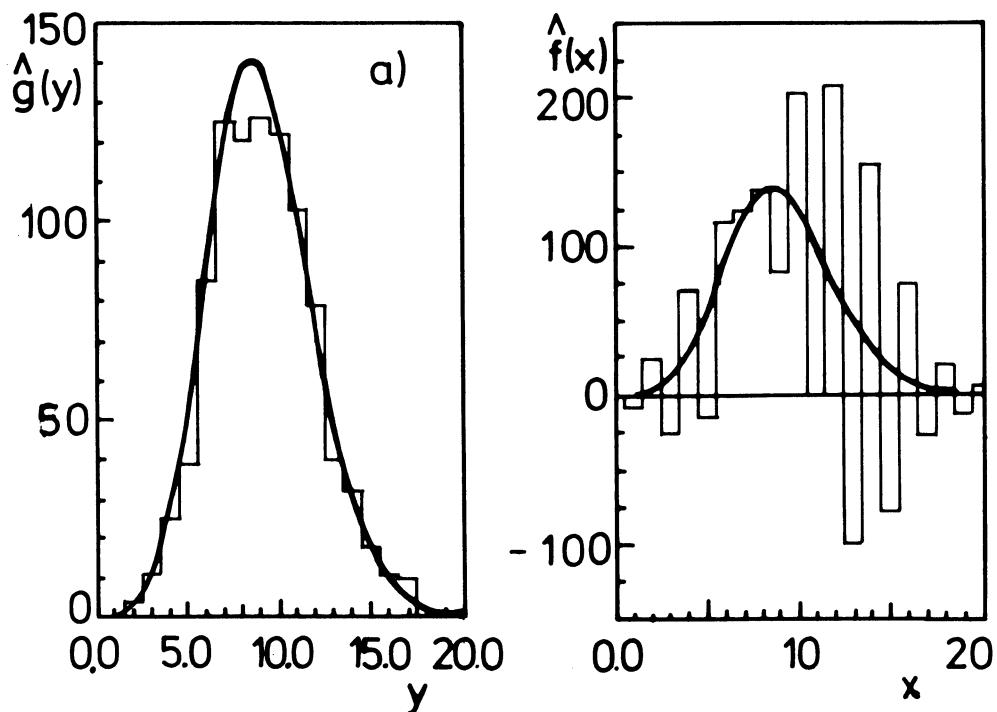


Figure 1. Distribution of the measured quantity y (a) and oscillating result of unfolding (b) using equation (1.05), shown as histograms. The original dependence is shown as a curve in both cases.

The probability to observe for y the true value x is 50 % (except for $x = 1$ and $x = n$). A certain dependence $f(x)$ is assumed and the expected distribution $g(y)$ is calculated by $g = Af$; a random sample of 1000 independent measurements of the variable y is generated. The result of the simulation is shown in Figure 1a as a histogram. Equation (1.07) is used to obtain an estimate \hat{f} for the original distribution, and this is

shown as a histogram in Figure 1b together with the original distribution (smooth curve). The result has an oscillating behaviour, and the covariance matrix shows large negative correlations, especially between adjacent values, although the effect of the resolution on the shape of the measured distribution appears to be rather small. The result of this example is typical for the direct solution of the unfolding problem, based on equation (1.07). Clearly this method is not acceptable.

In several experiments the factor method, as discussed above for the case of a pure acceptance correction, is used in the case of limited resolution as well. However now because of $A_{ij} \neq 0$ for $i \neq j$ the correction factor according to formula (1.06) will depend on the MC input histogram \bar{f} , and will only be correct, if \bar{f} is the true histogram. Since this is not known (otherwise the measurement would not be necessary), some assumption is necessary and the corrected values \hat{f}_j will be biased towards the MC input histogram. Often in applications of this kind, the MC input histogram is adjusted in several iterations, to get a histogram \bar{g} , which describes the measured histogram \hat{g} well. The essential point in applications of this method is to use always a smoothed histogram \bar{f} , otherwise the repeated application of equation (1.06) can be shown to give (in case of convergence) the result of the inversion method (which exactly reproduces \hat{g}). In general one can say, that the factor method applied to the case of finite resolution gives a biased result, with the danger to underestimate statistical and systematic uncertainties. Quite often the corrected result is presented in the form of a histogram with many bins, which are narrower than the resolution, and which cannot be resolved by the detector.

Acceptable unfolding results can be obtained by regularization methods [10] [11], which suppress the spurious oscillatory component in the solution. Regularization can be interpreted as the use of certain a-priori information on the degree of smoothness of the true solution. Since this can introduce a bias, the weight of the a-priori information has to be determined by statistical methods in order to keep any possible bias small compared to statistical errors. One unavoidable consequence of acceptable unfolding is the limitation of the number of unfolded data points, according to the resolution and the statistical accuracy. A measurement with limited resolution always means a loss of statistical accuracy.

A special feature of high-energy physics experiments is the fact that the response function is usually known only implicitly by the simulation of particle reactions in the detector. Limited statistical accuracy of the often very time-consuming Monte-Carlo calculations can introduce systematic uncertainties. The usual requirement 'number of MC events \approx number of measured events' is insufficient. Even more difficult, the Monte-Carlo simulation may require some assumptions, which have to be tested by comparison with the data, and therefore unfolding methods should allow sensitive tests of the assumptions. In addition the discretization has to be done rather carefully to avoid a further deterioration of the already limited resolution.

In this paper a general unfolding method, based on fundamental statistical principles is discussed in detail together with numerical examples. The method is similar to methods, used in other fields of science [12] [13] [14] and allows the inclusion of certain a priori information (regularization). It has been applied to several high-energy physics

experiments in an earlier [6] [15] and in the present version [7] [8]. The discussion is restricted to the case of a one-dimensional variable x and its distribution; the measured variable however may be multidimensional.

The description and discussion of the unfolding method is preceded by two chapters which introduce the basic concepts of statistics including parameter estimation and the parametrization of functions by spline and orthogonal functions.

2. STATISTICS

2.1 ONE-DIMENSIONAL RANDOM VARIABLES

The result of a measurement can be characterized by one or more real numbers x_i , $i = 1 \dots$. The probability that an experiment yields a result $a \leq x < b$ is given by

$$P(a \leq x < b) = \int_a^b f(x) dx \quad (2.01)$$

for a continuous random variable (r.v.) x , where $f(x)$ is the probability density function of the variable x . A probability density function is a nonnegative function with unit integral:

$$f(x) \geq 0 \quad \text{and} \quad \int_{-\infty}^{\infty} f(x) dx = 1. \quad (2.02)$$

Physicists usually call a probability density function (p.d.f.) a distribution, in statistics this name is reserved for the integrated p.d.f., which may be called cumulative distribution $F(x)$:

$$F(x) = \int_{-\infty}^x f(x') dx' \quad (2.03)$$

with $F(-\infty) = 0$ and $F(\infty) = 1$. An important parameter characterising the location of a random variable x is the expectation value of x , denoted by $E(x)$ and defined by

$$E(x) = \int_{-\infty}^{\infty} x f(x) dx. \quad (2.04)$$

Generalized for an arbitrary function $h(x)$, the expectation value of $h(x)$ is defined by

$$E(h) = \int_{-\infty}^{\infty} h(x) f(x) dx. \quad (2.05)$$

The expectation values of x^n and of $(x - E(x))^n$ are called n-th algebraic moments μ_n and n-th central moments μ'_n , respectively. The expectation value of x , or mean value μ of x , is equal to the first algebraic moment μ_1 ,

$$\mu = \mu_1 = E(x). \quad (2.06)$$

The second central moment μ'_2 is a measure of the spread of the distribution, it is called variance $V(x)$ and its definition is

$$V(x) = E((x - E(x))^2). \quad (2.07)$$

The variance is abbreviated by σ^2 , and σ is called the standard deviation.

The normal distribution. The normal or gaussian distribution is in practice the most important distribution, since measurement errors often follow the normal distribution. The density is

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \quad x \in (-\infty, \infty). \quad (2.08)$$

The normal distribution has two parameters μ and σ with $E(x) = \mu$ and $V(x) = \sigma^2$. The probability for x to fall into the region $\mu \pm \sigma$ is 68.3 %. The normal distribution for the case $\mu = 9$ and $\sigma = 3$ is shown in Figure 2 as a curve.

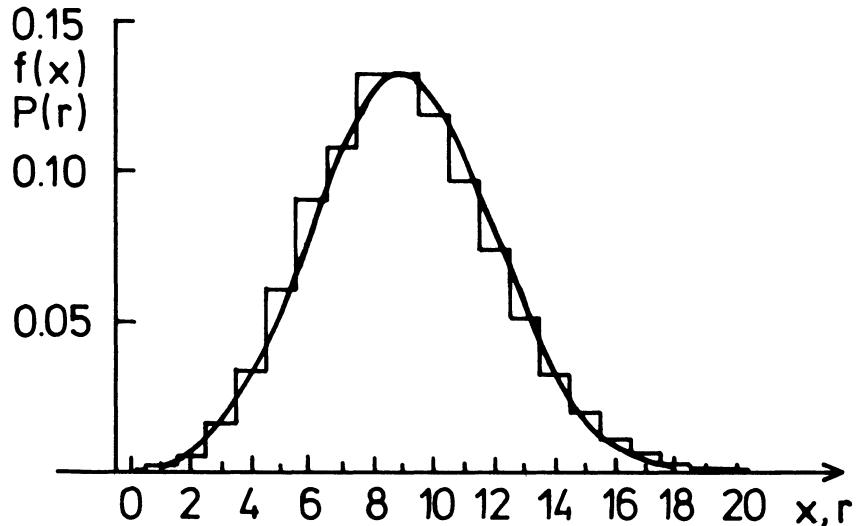


Figure 2. Normal distribution with $\mu = 9$ and $\sigma = 3$, shown as curve, and Poisson distribution with $\mu = 9$, shown as histogram.

A linear function $y = ax + b$ of a random variable x is again a random variable. The expectation value and the variance are

$$\mu_y = a\mu_x + b \quad \text{and} \quad \sigma_y^2 = a^2\sigma_x^2. \quad (2.09)$$

For a general transformation $y = h(x)$ the p.d.f. of y is

$$g(y) = \int_{-\infty}^{\infty} \delta(y - h(x))f(x) dx = \frac{f(x)}{|h'(x)|}, \quad (2.10)$$

if the function $y = h(x)$ is a one-to-one function.

In the case of a discrete variable r possible values can always be represented by a set of consecutive integers, $r \in \{a, a+1, a+2 \dots, b-1, b\}$. The probability, that an experiment yields the result r is denoted by the nonnegative number $P(r)$. Expectation values are defined in analogy to the case of continuous random variables, replacing integrals by sums.

The Poisson distribution. If events occur at a constant rate, the probability of observing in a given time interval exactly r events, is given by the Poisson distribution. For a Poisson distribution,

$$P(r|\mu) = e^{-\mu} \frac{\mu^r}{r!} \quad r \in \{0, 1 \dots \infty\} \quad (2.11)$$

represents the probability of observing r events, if the mean value is μ . The Poisson distribution has only one parameter μ , with $E(r) = \mu$ and $V(r) = \mu$. The Poisson distribution with $\mu = 9$ is shown in Figure 2 as a histogram. A comparison shows, that

for not too small mean values (say $\mu \geq 5$) the normal distribution represents a good approximation to the Poisson distribution, except in the tails.

2.2 MULTIDIMENSIONAL RANDOM VARIABLES

For a twodimensional random variable with components x and y the probability to observe (x, y) with $a \leq x < b$ and $c \leq y < d$ is

$$P(a \leq x < b, c \leq y < d) = \int_c^d \int_a^b f(x, y) dx dy \quad (2.12)$$

with the twodimensional p.d.f. $f(x, y)$, which obeys the normalization condition

$$\iint f(x, y) dx dy = 1 \quad (2.13)$$

with integration limits $-\infty$ and $+\infty$. Projections of the distribution $f(x, y)$ are called marginal distributions:

$$f_x(y) = \int_{-\infty}^{\infty} f(x, y) dx \quad f_y(x) = \int_{-\infty}^{\infty} f(x, y) dy. \quad (2.14)$$

They represent the distributions of the r.v. x and y , resp., if the other variable ignored. Sections through distributions $f(x, y)$ are called conditional distributions. Given a fixed value $x = x_0$, the conditional p.d.f. of y is

$$p(y|x_0) = \frac{f(x_0, y)}{\int f(x_0, y) dy} = \frac{f(x_0, y)}{f_y(x_0)}. \quad (2.15)$$

From the marginal p.d.f $f_y(x)$ and the conditional density $p(y|x)$ the joint p.d.f. is given by

$$f(x, y) = p(y|x) f_y(x) \quad (2.16)$$

and the marginal p.d.f. $h(y)$ is given by

$$f_x(y) = \int_{-\infty}^{\infty} p(y|x) f_y(x) dx. \quad (2.17)$$

This equation is similar to the basic equation (1.01); the difference is in the normalization. While the distributions defined here are always normalized, the distributions appearing in equation (1.01) are for example cross sections and the integrals are true or measured total cross sections.

The definitions of expectation values and variances are straightforward generalized to multidimensional r.v.:

$$\begin{aligned} E(x) &= \iint x f(x, y) dx dy & V(x) &= \iint (x - E(x))^2 f(x, y) dx dy \\ E(y) &= \iint y f(x, y) dx dy & V(y) &= \iint (y - E(y))^2 f(x, y) dx dy. \end{aligned} \quad (2.18)$$

For a two dimensional p.d.f. there is an additional characteristic, the covariance

$$\sigma_{xy} = cov(x, y) = \iint (x - E(x))(y - E(y)) f(x, y) dx dy. \quad (2.19)$$

The covariance can be expressed as $\sigma_{xy} = \rho_{xy} \sqrt{\sigma_x^2 + \sigma_y^2}$ with a correlation coefficient ρ_{xy} , which can take values between -1 and +1. The variables x and y are called uncorrelated, if $\rho_{xy} = 0$.

The p.d.f. of a random n -vector x , with components $x_i, i = 1 \dots n$ may be written as $f(x)$. As a generalization of the variance one can define a covariance matrix by

$$V = V(x) = E((x - E(x))(x - E(x))^T), \quad (2.20)$$

which is a symmetric $n \times n$ matrix. The diagonal elements $V(x_i) = \sigma_i^2$ are called variances, the off-diagonal elements $V(x_i, x_j) = \sigma_{ij}$ are called covariances. When parameter values are quoted as the result of an experiment, usually only the square roots of the diagonal elements are given as parameter errors. The confidence level for multidimensional regions in parameter space depends however on the covariances of the parameters [16].

The n -dimensional normal distribution. The p.d.f. of the n -dimensional normal (or gaussian) distribution depends on the n means, and on the n^2 elements of the covariance matrix V (with $(n^2 + n)/2$ different elements):

$$f(x) = \frac{1}{(2\pi)^{n/2}|V|^{1/2}} \exp(-\frac{1}{2}(x - \mu_x)^T V^{-1} (x - \mu_x)) \quad (2.21)$$

The n -dimensional gaussian distribution is the simplest model for the p.d.f. of n correlated random variables, since the only parameters are the n means and the $(n^2 + n)/2$ different elements of the covariance matrix.

The χ^2 distribution. If $x_1 \dots x_n$ are independent variables, which all follow the normal distribution with mean 0 and variance 1, the sum u of the squares

$$u = \sum_{i=1}^n x_i^2 \quad (2.22)$$

follows the χ^2 distribution $\chi^2(n)$ with n degrees of freedom. The probability density is given by

$$f(u) = \frac{\frac{1}{2}(\frac{u}{2})^{n/2-1} e^{-u/2}}{\Gamma(\frac{n}{2})}. \quad (2.23)$$

The expectation value is n and the variance is $2n$. The χ^2 distribution is important for statistical tests; tables are given in standard textbooks on statistics. The 95 % confidence level of the χ^2 distribution with one degree of freedom for example is 3.84. If x is a single variable, distributed normally with mean 0 and standard deviation σ , a measured value $(\hat{x}/\sigma)^2$ will be ≤ 3.84 with 95 % probability. Thus a measured value $(\hat{x}/\sigma)^2 \leq 3.84$ is compatible with zero in the 95 % confidence limit.

Linear functions of random variables. A linear transformation $y = Bx$ of a random n -vector x with mean μ_x and covariance matrix $V(x)$ to a m -vector y is considered. The expectation value μ_y of y and the covariance matrix $V(y)$ of y are given by:

$$\mu_y = B\mu_x \quad \text{and} \quad V(y) = BV(x)B^T, \quad (2.24)$$

where B^T is the matrix transposed to the matrix B . The expression for $V(y)$ is the equation of standard error propagation.

2.3 PARAMETER ESTIMATION

The estimation of parameters from measured data is a standard problem in data analysis. The application of the maximum likelihood method for a certain class of problems is discussed below.

Assume, that the dependence of a cross section $f(x)$ on a variable x has been measured. There exists a model, which expresses $f(x)$ as a sum of m known functions $p_j(x)$ with coefficients a_j :

$$f(x) = \sum_{j=1}^m a_j p_j(x). \quad (2.25)$$

The m parameters (or coefficients) have to be determined from the data, which are given in form of histogram bin contents $\hat{f}_i, i = 1 \dots n$. Each bin of width Δ is centered at a certain value x_i . Neglecting the variation of the functions $p_j(x)$ within a bin, the expected content of a histogram bin is

$$f_i = \Delta \cdot f(x) = \Delta \cdot \sum_{j=1}^m a_j p_j(x_i) = \sum_{j=1}^m A_{ij} a_j \quad \text{with} \quad A_{ij} = \Delta \cdot p_j(x_i). \quad (2.26)$$

The observed content of a histogram bin will follow a certain probability distribution, with a probability $P(\hat{f}_i|f_i)$ of observing \hat{f}_i , if the mean value is f_i . The product of all n probabilities,

$$L(a) = \prod_{i=1}^n P(\hat{f}_i|f_i) \quad (2.27)$$

is a function of the values of the parameters, and is called likelihood function. According to the maximum likelihood method [16], the best estimates of the parameters a are given by those values \hat{a} , for which the likelihood function takes on its largest value.

In applications of the maximum likelihood method usually a search is made for the minimum of the negative logarithm of the likelihood function:

$$S(a) = - \sum_{i=1}^n \ln P(\hat{f}_i|f_i). \quad (2.28)$$

Since the number of entries in a histogram bin will follow the Poisson distribution, the corresponding expression for $P(\hat{f}_i|f_i)$ can be inserted; dropping all constant terms, one gets

$$S(a) = \sum_{i=1}^n f_i - \sum_{i=1}^n \hat{f}_i \ln f_i. \quad (2.29)$$

Methods for the determination of the minimum usually are based on the approximation of $S(a)$ by a quadratic function, then the minimum can be determined by standard matrix methods. The derivatives of $S(a)$ w.r.t. the parameters at an approximate solution \tilde{a} are given by:

$$\frac{\partial S}{\partial a_j} = \sum_{i=1}^n A_{ij} - \sum_{i=1}^n \hat{f}_i \frac{A_{ij}}{f_i} \quad \frac{\partial^2 S}{\partial a_j \partial a_k} = \sum_{i=1}^n \hat{f}_i \frac{A_{ij} A_{ik}}{f_i^2} \quad (2.30)$$

with f_i calculated using the approximate solution \tilde{a} . In matrix notation the quadratic approximation can be written in the form

$$S(a) = S(\tilde{a}) - (a - \tilde{a})^T h + \frac{1}{2}(a - \tilde{a})^T H(a - \tilde{a}), \quad (2.31)$$

where h and H with elements

$$h_j = -\frac{\partial S}{\partial a_j} \quad H_{jk} = \frac{\partial^2 S}{\partial a_j \partial a_k} \quad (2.32)$$

are the (negative) gradient and the Hessian of $S(a)$, respectively. The minimum of the quadratic approximation above is defined by the condition $\nabla S = 0$,

$$-h + H(a - \tilde{a}) = 0, \quad (2.33)$$

which is solved by

$$a_{app} = \tilde{a} + H^{-1}h. \quad (2.34)$$

Since the obtained result a_{app} is based on the approximation of the true function $S(a)$ at \tilde{a} , several iterations have to be performed; in each iteration the result of the previous iteration replaces \tilde{a} . Convergence may be assumed, if both the expected change of $S(a)$ in one iteration,

$$(\Delta S)_{exp} = -\frac{1}{2}(a_{app} - \tilde{a})^T h \quad (2.35)$$

and the actual change of $S(a)$ are small compared to 1. The method is stable, if each iteration includes a search for the minimum of the function

$$S(t) = S(\tilde{a} + t(a_{app} - \tilde{a})), \quad (2.36)$$

depending on one parameter t . The result of the minimum search,

$$a_{min} = \tilde{a} + t_{min}(a_{app} - \tilde{a}), \quad (2.37)$$

then replaces \tilde{a} for the next iteration. It can be shown, that (negative) log likelihood functions are in fact approximately quadratic function, at least near the solution; therefore the value t_{min} is usually close to 1 and convergence with result \hat{a} is reached within few iterations.

A simpler method for the solution of the problem is based on the approximation of the Poisson distribution by the gaussian distribution. Under the conditions mentioned

in chapter 2.1 the Poisson probability may be approximated by the value of the gaussian density with $\sigma_i^2 = f_i$. Using the additional approximation $\sigma_i^2 = \hat{f}_i$ the problem becomes easily solvable. Inserting for $P(\hat{f}_i|f_i)$ the expression for the gaussian density, $S(a)$ becomes

$$S(a) = \frac{1}{2} \sum_{i=1}^n \frac{(\hat{f}_i - f_i)^2}{\sigma_i^2} \quad \sigma_i^2 = \hat{f}_i, \quad (2.38)$$

which is (except for the trivial factor 1/2) the expression to be minimized according to the least squares principle [17]. In this case the derivatives become

$$\frac{\partial S}{\partial a_j} = - \sum_{i=1}^n A_{ij} \frac{(\hat{f}_i - f_i)}{\sigma_i^2} \quad \frac{\partial^2 S}{\partial a_j \partial a_k} = \sum_{i=1}^n \frac{A_{ij} A_{ik}}{\sigma_i^2} \quad (2.39)$$

In matrix notation the function $S(a)$ can be written in the form

$$S(a) = S(\tilde{a}) - (a - \tilde{a})^T h + \frac{1}{2}(a - \tilde{a})^T H(a - \tilde{a}), \quad (2.40)$$

identical to the Poisson case and therefore can be treated in the same way. However, since $S(a)$ in this case is really quadratically in a , no iteration is necessary and the result

$$\hat{a} = H^{-1} h \quad (2.41)$$

is obtained in one step, which does not require an approximate starting value \tilde{a} . Thus the least squares method may be used to calculate an approximate solution, required by the method based on the Poisson distribution.

Since \hat{a} of equation above is a linear function of the gradient h , which itself is a linear function of the measured data \hat{f}_i , the simple formula of error propagation can be used to calculate the covariance matrix $V(\hat{a})$. In order to derive the result with matrix methods, a n -by- n weight matrix W is introduced, which is the inverse of the covariance matrix $V(\hat{f})$ of the measured data. Since the data are uncorrelated, the matrix $V(\hat{f})$ is a diagonal matrix with diagonal elements σ_i^2 , and the weight matrix W is diagonal too, with diagonal elements $1/\sigma_i^2$. The (negative) gradient h and the Hessian H are given by the matrix expressions

$$h = A^T W \hat{f} \quad H = A^T W A, \quad (2.42)$$

and the full solution can be written in the form

$$\hat{a} = H^{-1} h = (A^T W A)^{-1} A^T W \hat{f}. \quad (2.43)$$

Using the formula of error propagation, the covariance matrix $V(\hat{a})$ is

$$V(\hat{a}) = (A^T W A)^{-1} A^T W W^{-1} W A (A^T W A)^{-1} = (A^T W A)^{-1} = H^{-1}. \quad (2.44)$$

It can be shown that the same simple formula $V(\hat{a}) = H^{-1}$ applies to the result, obtained with the maximum likelihood method based on the Poisson distribution. This is at least true for the asymptotic case of high statistics, but is a good approximation already for low statistics.

3. PARAMETRIZATION OF FUNCTIONS

3.1 INTERPOLATING SPLINE FUNCTIONS

Spline functions [18] are smooth interpolating functions. Besides applications in graphics, spline functions are increasingly used for numerical problems, for example in methods of solving boundary-value problems of differential equations. The standard application of spline functions is the interpolation between pairs (y_i, x_i) with $x_i \in [a, b]$. The set $Y = \{y_0, y_1 \dots y_n\}$ of $(n + 1)$ real numbers represents values of a function $f(x)$ at abscissa values x_i , called knots. The set $X = \{x_0, x_1 \dots x_n\}$ with $x_0 = a$ and $x_n = b$ can be considered as a partition of the interval $[a, b]$. A cubic spline function $S(x)$ with $S(x_i) = y_i$, $i = 0 \dots n$, is a twice continuously differentiable function on $[a, b]$, coinciding on every subinterval $[x_i, x_{i+1}]$, $i = 0 \dots (n - 1)$ with a polynomial of third degree:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3. \quad (3.01)$$

At every inner knot x_i the two polynomials of the adjacent subintervals agree in the function values and in the values of the first two derivatives.

A cubic spline function with total $4n$ coefficients is not uniquely defined. The requirements $S(x_i) = y_i$ give $(n + 1)$ conditions for the coefficients, and with the $(n - 1)$ conditions at the inner knots on $S(x)$, $S'(x)$ and $S''(x)$, there are in total $(n + 1) + 3(n - 1) = 4n - 2$ conditions for the $4n$ coefficients. Thus two degrees of freedom are left, which have to be fixed by two additional conditions. Often used conditions are the following:

- (I) $S''(x_0) = 0$; $S''(x_n) = 0$ (natural spline)
- (II) $S'(x_0) = y'_0$; $S'(x_n) = y'_n$ (complete spline).

A natural requirement for an interpolating function is a certain degree of smoothness. Since the local curvature $f''(x)/(1 + f'^2(x))^{3/2}$ of a function $f(x)$ can be approximated for $|f'(x)| \ll 1$ by f'' , the integral

$$\int_a^b [f''(x)]^2 dx \quad (3.02)$$

appears to be a reasonable quantitative measure of the smoothness of a function $f(x)$. This quantity will be called the (total) curvature of a function $f(x)$ in an interval $[a, b]$ in the following.

The natural spline (condition (I) above) is the smoothest function to interpolate given support points (y_i, x_i) in the sense of the curvature (3.02). First a proof is given for the statement:

$$0 \leq \int_a^b |f''(x) - S''(x)|^2 dx = \int_a^b |f''(x)|^2 dx - \int_a^b |S''(x)|^2 dx \quad (3.03)$$

for cubic spline functions $S(x)$ under conditions (I) and (II), and twice continuously differentiable functions $f(x)$. The first integral of equation (3.03) can be rewritten in

the form

$$\int_a^b |f''(x) - S''(x)|^2 dx = \int_a^b |f''(x)|^2 dx - \int_a^b |S''(x)|^2 dx - 2 \int_a^b (f''(x) - S''(x))S''(x) dx \quad (3.04)$$

The last term is integrated by parts. For each subinterval $[x_{i-1}, x_i]$, $i = 1 \dots n$,

$$\begin{aligned} \int_{x_{i-1}}^{x_i} (f''(x) - S''(x))S''(x) dx &= (f'(x) - S'(x))S''(x) \Big|_{x_{i-1}}^{x_i} \\ &\quad - \int_{x_{i-1}}^{x_i} (f'(x) - S'(x))S^{(3)}(x) dx \\ &= (f'(x) - S'(x))S''(x) \Big|_{x_{i-1}}^{x_i} - (f(x) - S(x))S^{(3)}(x) \Big|_{x_{i-1}}^{x_i} \\ &\quad + \int_{x_{i-1}}^{x_i} (f(x) - S(x))S^{(4)}(x) dx \end{aligned} \quad (3.05)$$

And adding up all terms one gets with $S^{(4)}(x) = 0$

$$\begin{aligned} 0 \leq \int_a^b |f''(x) - S''(x)|^2 dx \\ &= \int_a^b |f''(x)|^2 dx - \int_a^b |S''(x)|^2 dx - 2(f'(x) - S'(x))S''(x) \Big|_a^b \\ &\quad + 2 \sum_{i=1}^n (f(x) - S(x))S'''(x) \Big|_{x_{i-1}}^{x_i} \end{aligned} \quad (3.06)$$

The last term (the sum) vanishes because of the interpolation condition $S(x_i) = y_i$, $i = 0 \dots n$, and the term before vanishes under conditions (I) and (II). Thus the inequality

$$\int_a^b |S''(x)|^2 dx \leq \int_a^b |f''(x)|^2 dx, \quad (3.07)$$

follows and proves the statement (3.02). The inequality (3.07) implies that, among all twice continuously differentiable functions $f(x)$ with $f(x_i) = x_i$ the spline function $S(x)$ with condition (I) (natural spline) minimizes the total curvature (3.03). It also minimizes approximately the 'strain energy' in a curved elastic bar

$$\int_a^b \frac{(f''(x))^2}{(1 + f'(x))^{5/2}} dx \quad (3.08)$$

and this property is the origin of the name 'spline'. As an example the interpolation of 10 given points (y_i, x_i) by a spline function is shown in Figure 3, and is compared with the interpolation by a polynomial (of degree 9). The polynomial shows large oscillations near the endpoints, typical for the interpolation of equidistant data by a high order polynomial [19]. The advantages of the spline interpolation are apparent.

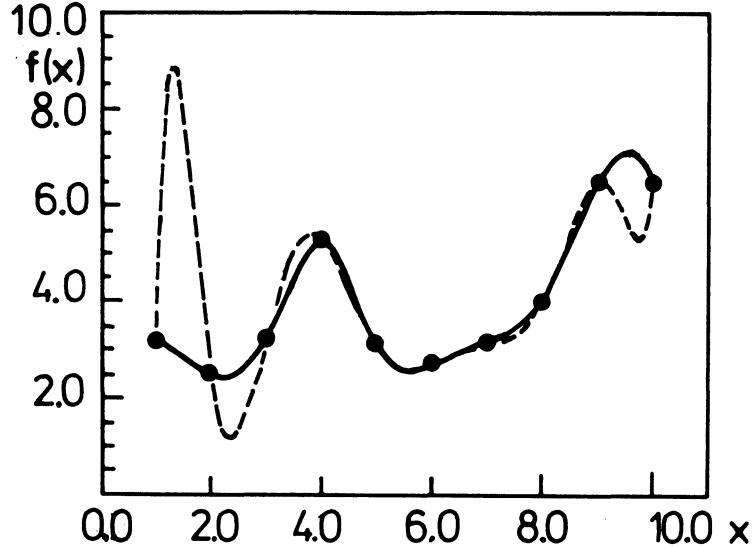


Figure 3. Interpolation of 10 given points (y_i, x_i) by a cubic spline function (full curve) and by a ninth order polynomial (dashed curve).

However, smoothness is not the only criterion for an interpolating function. A more important criterion is the accuracy of approximation of a given function $f(x)$ by $S(x)$. From an approximation point of view, the properties of spline functions with condition (I) are not optimal (unless really $f''(x_0) = f''(x_n) = 0$). If the true slopes at the end points are known, condition (II) gives a better approximation, and if the values of the second derivatives are known at the end points, the condition

$$(III) \quad S''(x_0) = y_0'' \quad ; \quad S''(x_n) = y_n''$$

can be used. If nothing is known about end point derivatives, one can use as a general condition

$$(IV) \quad S'''(x) \text{ continuous across } x_1 \text{ and } x_{n-1} \quad (\text{not-a-knot condition}).$$

The not-a-knot condition means, that the first and last inner knots are not active.

Spline functions $S(x)$ have optimal approximation properties (for a quantitative treatment see [18]). The difference $|f(x) - S(x)|$ is bounded by a quantity proportional to the fourth power of the knot spacing. Even the k -th derivatives of $f(x)$ for $k = 1, 2$ and 3 are well approximated (with a bound on the difference to the true derivative proportional to the $(4 - k)$ -th power of the knot spacing). The natural spline condition (I) introduces an error proportional to the square of the knot spacing near the ends, which is avoided by the not-a-knot condition (IV) and therefore this condition seems to be preferable as a general condition. In fact the not-a-knot condition has been used for the construction of the interpolating spline in Figure 3.

The determination of spline functions $S(x)$ for given support points (y_i, x_i) is a stable process for all conditions (I) - (IV); algorithms for the determination of the coefficients $a_i, b_i, c_i, d_i, i = 1 \dots (n - 1)$ can be found in [18], [20]. The construction of

higher order spline functions is possible, in practice however there is rarely a reason to go beyond cubic spline functions.

3.2 B-SPLINES

A representation of spline functions $S(x)$, different from the one given by equation (3.01), but equivalent, is provided by the so called basis splines or short B-splines [20]. A B-spline is itself a spline function. A single B-spline of order k is nonzero only in a limited range of x (basis). A spline function $S(x)$ of order k can be represented by a sum of B-splines $B_{j,k}(x)$ according to

$$S(x) = \sum_j a_j B_{j,k}(x), \quad (3.09)$$

where the functions $B_{j,k}(x)$ are B-splines of order k . The representation of spline functions by a linear combination of B-splines has numerical advantages in certain applications, for example least squares fits of a spline function to data become linear.

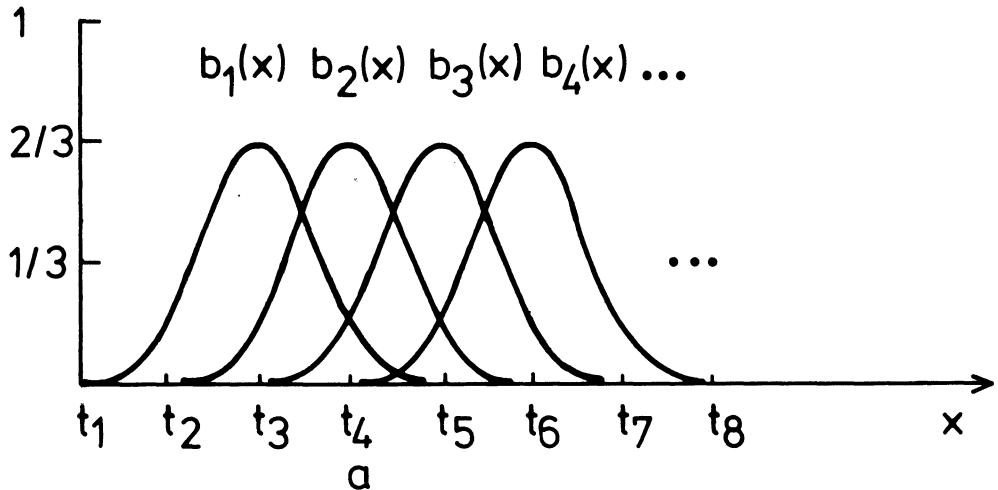


Figure 4. Sequence of cubic B-splines and equidistant knots.

B-splines are defined over a nondecreasing sequence $\{t_j\}$ of knots. Order $k = 1$ B-splines are defined by

$$B_{j,1} = \begin{cases} 1 & t_j \leq x < t_{j+1} \\ 0 & \text{otherwise.} \end{cases} \quad (3.10)$$

Higher order B-splines $B_{j,k}(x)$ (with $k > 1$) are defined by a recurrence relation, which allows to compute $B_{j,k}(x)$ by positive linear combinations of positive quantities:

$$B_{j,k} = \frac{x - t_j}{t_{j+k-1} - t_j} B_{j,k-1}(x) + \frac{t_{j+k} - x}{t_{j+k} - t_{j+1}} B_{j+1,k-1}(x). \quad (3.11)$$

B-splines $B_{j,k}(x)$ have the property

$$B_{j,k}(x) = \begin{cases} > 0 & t_j < x < t_{j+k}, \\ 0 & \text{otherwise} \end{cases}, \quad (3.12)$$

i.e. they take on only positive values (or zero). In any particular interval $[t_j, t_{j+1}]$ only the k B-splines $B_{j-k+1,k}(x) \dots B_{j,k}(x)$ may be nonzero. B-splines, as defined in equations (3.10) and (3.11), are normalized in the sense, that at any particular x -value their sum is equal to 1:

$$\sum_j B_{j,k}(x) = \sum_{j=l+1-k}^l B_{j,k}(x) = 1 \quad t_l \leq x < t_{l+1}. \quad (3.13)$$

In the following only cubic B-splines ($k = 4$) are considered. For the case of B-splines $B_{j,4}(x)$ with equidistant knots, which are often sufficient, explicit formulas are given, denoting these special B-splines by $b_j(x)$. If m B-splines $b_j(x)$ are used for the parametrization of a function for $a \leq x \leq b$, the distance between adjacent knots is $d = (b - a)/(m - 3)$. In total there are $(m + 4)$ knots, with $t_4 = a$, $t_{m+1} = b$ and in general $t_j = a + (j - 4)d$ (see Figure 4). The explicit formulas for the B-splines $b_j(x)$ in terms of a variable z with $0 \leq z < 1$ are:

$$b_j(z) = \begin{cases} \frac{1}{6}z^3 & z = (x - t_j)/d \\ \frac{1}{6}[1 + 3(1 + z(1 - z))z] & t_j \leq x < t_{j+1} \\ \frac{1}{6}[1 + 3(1 + z(1 - z))(1 - z)] & t_{j+1} \leq x < t_{j+2} \\ \frac{1}{6}(1 - z)^3 & t_{j+2} \leq x < t_{j+3} \\ 0 & t_{j+3} \leq x < t_{j+4} \\ \text{otherwise} & \end{cases} \quad (3.14)$$

A single B-spline $b_j(x)$ together with the derivatives is shown in Figure 5.

The m coefficients for a spline function in the parametrization (3.09) with B-splines $b_j(x)$, interpolating $(m - 2)$ equidistant data points (y_i, x_i) with $x_i = t_i$, $i = 4, \dots, m + 1$, are determined by a set of linear equations. The first $(m - 2)$ equations

$$6y_j = a_{j-3} + 4a_{j-2} + a_{j-1} \quad j = 4 \dots m + 1 \quad (3.15)$$

are given by the interpolation conditions. The additional two conditions given by the not-a-knot condition (see chapter 3.1) are

$$\begin{aligned} -a_1 + 4a_2 - 6a_3 + 4a_4 - a_5 &= 0 \\ -a_{m-4} + 4a_{m-3} - 6a_{m-2} + 4a_{m-1} - a_m &= 0. \end{aligned} \quad (3.16)$$

In a parametrization (3.09) by B-splines, each function value $S(x)$ at a certain x as well as derivatives $S'(x) \dots$ and the integral of $S(x)$ over a x -region is a linear combination of the coefficients a_j . This property allows linear least squares fits to data (y_i, x_i) of spline functions in the B-spline parametrization. Having determined the coefficients a_j together with their covariance matrix, error propagation for interpolated function values, derivatives etc. is straightforward.

3.3 ORTHOGONAL FUNCTIONS

The concept of orthogonal functions is of particular importance in many fields of numerical and statistical analysis. A system $\{p_j(x)\}$ of functions $p_j(x)$, defined for $a \leq x \leq b$, is called orthogonal, if the inner product of each two functions (p_j, p_k)

$$(p_j, p_k) = \int_a^b p_j(x)p_k(x) dx = 0 \quad \text{for } j \neq k. \quad (3.17)$$

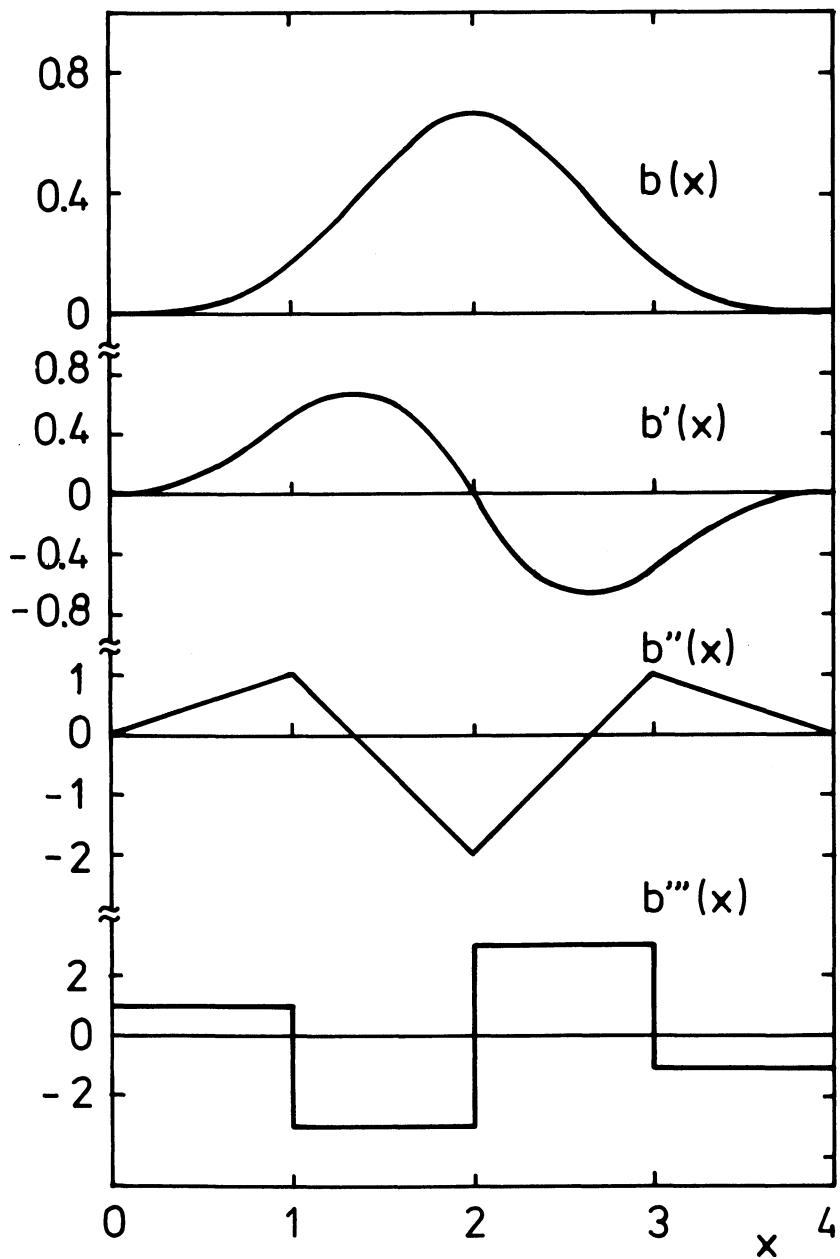


Figure 5. Single B-spline and derivatives, for equidistant knots with distance $d = 1$ between adjacent knots.

Orthogonal functions are called normalized, if the values N_j , defined by

$$\int_a^b p_j^2(x) dx = N_j \quad (3.18)$$

are equal to 1 for all values of j . By an orthogonalization procedure a system of orthogonal functions can be constructed from a system of linear independent functions [21]. A piecewise continuous function $g(x)$ can be expanded in terms of a system of normalized

orthogonal functions,

$$g(x) = \sum_{j=1}^{\infty} a_j p_j(x) \quad \text{with} \quad a_j = \int_a^b p_j(x) g(x) dx, \quad (3.19)$$

where the factors a_j are called expansion coefficients or components of the function $f(x)$ with respect to the system $\{p_j(x)\}$. The deviation between the function $f(x)$ and a finite sum with n terms can be measured by the quadratic expression

$$M_m = \int_a^b \left[f(x) - \sum_{j=1}^m a_j p_j(x) \right]^2 dx \quad (3.20)$$

According to the principle of least squares the quantity M_m should be as small as possible. It can be shown [21], that

$$\sum_{j=1}^m a_j^2 \leq \int_a^b [f(x)]^2 dx \quad (3.21)$$

A system of orthogonal functions is called complete, if for any positive number ϵ there is an index m , such that $M_m < \epsilon$. In practice usually only a small number of terms is necessary, since the magnitude of $|a_j|$ falls rapidly with increasing index j above a certain index. This feature allows an efficient representation of functions $f(x)$ by finite sums, which can be evaluated very fast, using recurrence relations for the evaluation of the $p_j(x)$ [19].

An important example of a complete orthogonal system of functions is given by the functions $1, \cos x, \sin x, \cos 2x, \sin 2x, \dots$, which is orthogonal in the interval $0 \leq x \leq 2\pi$. A periodic function, with period 2π , can be expanded in a Fourier sum

$$f(x) = \frac{a_0}{2} + \sum_{\nu=1}^{\infty} (a_{\nu} \cos \nu x + b_{\nu} \sin \nu x), \quad (3.22)$$

where the expansion coefficients a_{ν} and b_{ν} (Fourier coefficients) are given by the formulas:

$$a_{\nu} = \frac{1}{\pi} \int_0^{2\pi} f(x) \cos \nu x dx \quad b_{\nu} = \frac{1}{\pi} \int_0^{2\pi} f(x) \sin \nu x dx. \quad (3.23)$$

It can be shown, that the trigonometric interpolation (3.22) converges to the given function $f(x)$ at every point on the range [22]. The asymptotic behaviour of the Fourier coefficients depends on the degree of differentiability of $f(x)$:

$$\sqrt{a_{\nu}^2 + b_{\nu}^2} = O \left(\frac{1}{|\nu|^{r+1}} \right) \quad (3.24)$$

for a 2π -periodic function $f(x)$ having an absolutely continuous r -th derivative [18]. Another interesting property is the fact, that each term of the expansion represents an independent contribution to the total curvature:

$$\int_0^{2\pi} [f''(x)]^2 dx = \pi \sum_{\nu=1}^{\infty} \nu^4 (a_{\nu}^2 + b_{\nu}^2). \quad (3.25)$$

Equations (3.24) and (3.25) show, that for a smooth function $f(x)$ (for example $r \geq 3$ in equation (3.24)) the coefficients of the higher terms of the expansion, which have large contributions to the curvature, fall rapidly with increasing index ν . One method to smooth a set of empirical data (y_i, x_i) with statistical fluctuations is the determination of the Fourier coefficients (analysis), the attenuation of the coefficients with higher index representing noise only, and the reconstruction (synthesis) according to equation (3.22) [23].

A system of orthogonal polynomials can be constructed from the monomials 1, x , x^2 For the region $-1 \leq x \leq 1$ the orthogonal polynomials are identical to the Legendre polynomials (apart from constant factors). Orthogonal polynomials are often used to approximate an empirical function given by a discrete set of points (y_i, x_i) , $i = 1 \dots n$, if no specific parametrization is known. Following the least squares principle, orthogonal polynomials for the discrete set can be constructed from a recurrence relation, starting from a constant and a linear term, by the requirement

$$\sum_{i=1}^n w_i p_j(x_i) p_k(x_i) = \delta_{jk}, \quad (3.26)$$

where w_i is the weight of an individual data point, which can be defined by $w_i = 1/\sigma_i^2$ for the standard deviation σ_i [17]. For the normalized orthogonal polynomials $p_j(x)$ the coefficients a_j of the approximation

$$f(x) = \sum_j a_j p_j(x) \quad (3.27)$$

can be calculated by summation:

$$a_j = \sum_{i=1}^n w_i p_j(x_i) y_i. \quad (3.28)$$

Because of the orthogonality of the functions $p_j(x)$, the covariance matrix $V(a)$ of the coefficients is a unit matrix I (this corresponds to a least squares fit with $H = I$, compare chapter 2.3). This property allows statistical χ^2 tests on the significance of each coefficient a_j . If all coefficients a_j for $j > m_0$ are compatible with zero, the discrete set (y_i, x_i) can be approximated by m_0 terms of the expansion (3.27), the lowest order polynomial consistent with the data.

4. UNFOLDING OF CONTINUOUS DISTRIBUTIONS

4.1 UNFOLDING OF PERIODIC FUNCTIONS

In this chapter the difficulties inherent in unfolding procedures are discussed in a special case, which makes them clearly apparent. Consider the case of a function $f(x)$ in the range $0 \leq x \leq 2\pi$, periodic with the period 2π , which is measured with a gaussian resolution function.

Using the formulae (3.23) for the determination of the Fourier coefficients, a piecewise continuous function $f(x)$ can be expanded according to

$$f(x) = \frac{a_0}{2} + \sum_{\nu=1}^{\infty} (a_{\nu} \cos \nu x + b_{\nu} \sin \nu x) \quad (4.01)$$

with $a_{\nu}, b_{\nu} \rightarrow 0$ for $\nu \rightarrow \infty$. The folding by a gaussian resolution function with a standard deviation σ gives

$$g(y) = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y-x)^2}{2\sigma^2}\right) f(x) dx. \quad (4.02)$$

For a single term $\cos \nu x$ of the expansion (4.01) one gets

$$\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y-x)^2}{2\sigma^2}\right) \cos \nu x dx = \exp\left(-\frac{\nu^2\sigma^2}{2}\right) \cos \nu y \quad (4.03)$$

This means, that a single term in the expansion (4.01) has, after folding, the same form as before. The amplitude however is attenuated by a factor $\exp(-\nu^2\sigma^2/2)$, as shown in Figure 6 for two terms of the expansion (4.01).

This interesting result which is true for all $\cos \nu x$ and $\sin \nu x$ terms, shows how to unfold a measured periodic function. The measured function has to be expanded in the form

$$g(y) = \frac{a_0}{2} + \sum_{\nu=1}^{\infty} (\alpha_{\nu} \cos \nu x + \beta_{\nu} \sin \nu x) \quad (4.04)$$

by formulae which are equivalent to formulae (3.23). Unfolding and reconstruction of the original function $f(x)$ is then done by

$$a_{\nu} = \exp\left(\frac{\nu^2\sigma^2}{2}\right) \alpha_{\nu} \quad b_{\nu} = \exp\left(\frac{\nu^2\sigma^2}{2}\right) \beta_{\nu} \quad (4.05)$$

These exact formulae show very clearly the difficulties of unfolding. The coefficients α_{ν} and β_{ν} can only be determined with some statistical errors, while the true values become smaller with increasing value of ν ; the multiplication in formulae (4.05) then means the multiplication of the statistical errors with a rapidly increasing exponential factor, and the unfolded result would soon be dominated by statistical fluctuations. The reconstruction of the coefficients a_{ν}, b_{ν} above a certain value of the index ν becomes meaningless. This means that one either obtains very large unwanted fluctuations in

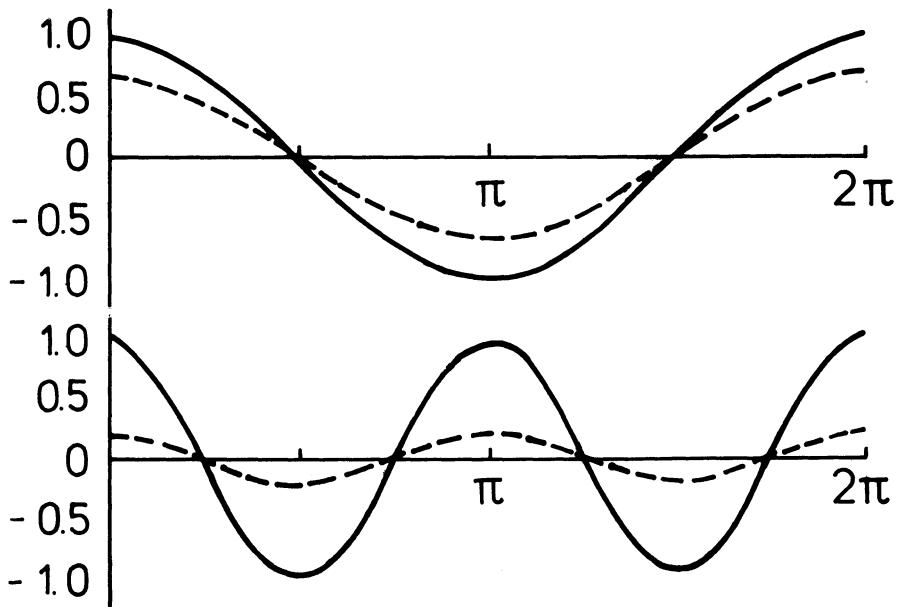


Figure 6. Graph of the functions $\cos x$ and $\cos 2x$ (full curves) and of the same functions after folding with a gaussian resolution function with $\sigma^2 = 3/4$.

the solution, or one has to limit the number of Fourier terms which prevents the finer structures to be resolved. The latter is of course expected as a consequence of the limited resolution.

4.2 DISCRETIZATION

The fundamental equation (1.01) relates the true distribution $f(x)$ to the distribution $g(y)$ measured in an experiment. In actual experiments the measured distribution usually contains some background contribution from other sources. It is assumed, that the background $b(y)$ can be either measured independently or calculated; in any case, the background is assumed to be known. Thus, for a given true distribution $f(x)$, the expected distribution $g(y)$ in the measured variables y can be written in the form:

$$g(y) = \int_a^b A(y, x) f(x) dx + b(y). \quad (4.06)$$

The distribution $\hat{g}(y)$ actually measured differs from the expected distribution $g(y)$ by some statistical errors $\epsilon(x)$. To obtain the true dependence $f(x)$ from the data, equation (4.06) has to be discretized, representing the continuous function $f(x)$ by a finite set of coefficients $a_1, a_2 \dots a_j \dots a_m$. The discretization, as described below, results in an equation of the form

$$g = Aa + b, \quad (4.07)$$

where g , a and b are vectors and A is a matrix, representing the response function $A(y, x)$. As remarked in chapter 1, acceptance and resolution in high-energy physics

experiments are usually defined only implicitly by MC-procedures and this fact has to be taken into account in the discretization method.

The discretization of equation (4.06) is done in two steps. In a first step the function $f(x)$ is parametrized by a sum

$$f(x) = \sum_{j=1}^m a_j p_j(x), \quad (4.08)$$

using a certain set of basis functions $p_j(x)$ to be specified later. The parametrization (4.08) allows to perform the integration:

$$\int_a^b A(y, x) f(x) dx = \sum_{j=1}^m a_j \left[\int_a^b A(y, x) p_j(x) dx \right] = \sum_{j=1}^m a_j A_j(y)$$

with $A_j(y) = \int_a^b A(y, x) p_j(x) dx.$

(4.09)

Now equation (4.06) can be rewritten in the form

$$g(y) = \sum_{j=1}^m a_j A_j(y) + b(y). \quad (4.10)$$

The expected distribution $g(y)$ is expressed by a superposition of functions $A_j(y)$, each representing one term $p_j(x)$ in the representation (4.08).

The second discretization step is the representation of all y -dependent functions in equation (4.10) by histograms, assuming a certain set of bin-limits $y_0, y_1 \dots y_n$:

$$g_i = \int_{y_{i-1}}^{y_i} g(y) dy \quad A_{ij} = \int_{y_{i-1}}^{y_i} A_j(y) dy \quad b_j = \int_{y_{i-1}}^{y_i} b(y) dy. \quad (4.11)$$

Using this discretization equation (4.06) can be written in the form of equation (4.07). g and b are n -vectors, representing histograms of the measured quantity y . The vector a is a m -vector of coefficients a_j , and A is a n -by- m matrix of elements A_{ij} ; column A_j of matrix A represents the histogram in y for $f(x) = p_j(x)$. The elements A_{ij} of matrix A are defined by the MC-events. Each MC-event, with true value x , is added to histogram $A_j(y)$ with a weight proportional to $p_j(x)$. In order to avoid negative weights and to simplify a proper normalization, the conditions

$$p_j(x) \geq 0 \quad \sum_{j=1}^m p_j(x) \equiv 1 \quad (4.12)$$

are required. That is, the sum of all weights $p_j(x)$ for a given MC-event is equal to 1. In addition an overall weight for the MC-events has to be defined, such that the resulting distribution $f(x)$ is correctly normalized. In particle reactions, where $f(x)$ is a cross section, the ratio of event number to cross section is called integrated luminosity $\int L dt$:

$$N(x) = f(x) \int L dt. \quad (4.13)$$

The integrated luminosity of the experiment has of course to be known, either from the experimental conditions or by the measurement of a monitor reaction. For the MC event simulation one can define a luminosity as well, by

$$\int L_{MC} dt = \frac{N_{MC}}{\int f_0(x) dx}, \quad (4.14)$$

if the MC events are generated according to $f_0(x)$. The simplest way is to use an unit cross section $f_0(x) \equiv 1$. If in this case the ratio of the integrated luminosities

$$\frac{\int L_{exp} dt}{\int L_{MC} dt} \quad (4.15)$$

is used as an overall weight of the MC events, the resulting $f(x)$ will be directly the correctly normalized cross section.

Some remarks are in order concerning the discretization. The choice of the basis function $p_j(x)$ is an important point. The simplest choice compatible with the conditions of eq.(4.12) is :

$$p_j(x) = \begin{cases} 1 & \text{for } t_{j-1} \leq x < t_j \\ 0 & \text{otherwise} \end{cases} \quad (4.16)$$

with a set of knots $t_0, t_1, t_2 \dots t_m$. This choice has the property, that the coefficients a_j directly represent a histogram of the solution $f(x)$. However, the function $f(x)$ would have discontinuities, and the step function approximation of the integrand in eqation (4.09) is inaccurate. An obvious choice for the basis functions $p_j(x)$ are cubic B-splines $b_j(x)$, which satisfy the requirements of eqation (4.12) and provide an accurate integration. Note, that the step functions defined in eqation (4.16) are also B-splines, of order $k = 0$, while cubic B-splines are of order $k = 4$. Besides the accurate integration cubic B-splines have further advantages, to be discussed later. Using cubic B-splines, the solution $f(x)$ is a smooth curve, determined by the m coefficients a_j . The final result should of course be represented by data points with error bars. These data points can be obtained by integration of $f(x)$ over small regions in x ,

$$f_k = \left(\int_{x_{k-1}}^{x_k} f(x) dx \right) / (x_k - x_{k-1}) = \left(\sum_{j=1}^m a_j \int_{x_{k-1}}^{x_k} p_j(x) dx \right) / (x_k - x_{k-1}), \quad (4.17)$$

with the values f_k representing average values of the function $f(x)$ in $[x_{k-1}, x_k]$. Since the values f_k are linear functions of the coefficients a_j , the calculation of errors is straightforward.

In general it will be more efficient to generate MC event according to a distribution $f_0(x)$, which is close to the true distribution $f(x)$. The same definition of the integrated MC luminosity may be used as before, but the resulting $f(x)$ has to be multiplied by $f_0(x)$.

4.3 UNFOLDING WITHOUT REGULARIZATION

The discretization derived in the preceeding chapter facilitates unfolding through a fit of the linear expression $g = Aa + b$ to the data \hat{g} . The problem belongs to the class of

problems discussed in chapter 2.3. The solution \hat{a} obtained in the unfolding case usually looks unsatisfactory: the resulting distribution

$$\hat{f}(x) = \sum_{j=1}^m \hat{a}_j p_j(x) \quad (4.17)$$

can show an oscillating behaviour, as mentioned in chapter 1 and discussed quantitatively in chapter 4.1, with fluctuations much larger than any physically motivated expectations.

In order to analyse the reason for this unwanted behaviour in detail, the solution of the problem is derived in a different way. One iteration step of the algorithm of chapter 2.3 is considered, which consists in the determination of the minimum of the quadratic approximation of the negative log likelihood function, starting from a previous estimate \tilde{a} :

$$S(a) = S(\tilde{a}) - (a - \tilde{a})^T h + \frac{1}{2}(a - \tilde{a})^T H(a - \tilde{a}). \quad (4.18)$$

In the following the parameters are transformed to a new basis. Since the matrix H is symmetric, it can be transformed to a diagonal matrix D ,

$$D = U_1^T H U_1, \quad (4.19)$$

where the matrix D contains in its diagonal the real eigenvalues D_{jj} of H , which are positive because the matrix H is positive definite; the matrix U_1 is an orthogonal matrix with the property $U_1^T U_1 = I_{mm}$ and contains the eigenvectors u_j corresponding to the eigenvalues in its columns. The eigenvalues may be arranged in decreasing order $D_{11} \geq D_{22} \geq \dots \geq D_{mm}$; in typical applications they decrease by several orders of magnitudes. A diagonal matrix $D^{1/2}$ with the property $D^{1/2} D^{1/2} = D$ can be defined, which has the positive square roots of D_{jj} in the diagonal. A transformation is defined between the parameter vector a and another vector a_1 by

$$a = U_1 D^{-1/2} a_1. \quad (4.20)$$

Inserting this expression into equation (4.18) one gets after omitting constant terms,

$$S(a_1) = -a_1^T D^{-1/2} U_1^T (H\tilde{a} + h) + \frac{1}{2} a_1^T a_1. \quad (4.21)$$

From the minimum condition $\nabla S = 0$ the solution

$$\hat{a}_1 = D^{-1/2} U_1^T (H\tilde{a} + h) \quad (4.22)$$

is obtained directly. The remarkable feature of the parameters \hat{a}_1 , achieved by the transformation, is, that the covariance matrix $V(\hat{a}_1)$ is equal to the unit matrix I . This means, that the different components $(\hat{a}_1)_j$ of \hat{a}_1 are statistically independent and have a variance of 1. The result obtained is of course equivalent to the solution derived in chapter 2.4, which can be shown by a transformation back to a :

$$\hat{a} = U_1 D^{-1/2} \hat{a}_1 = U_1 D^{-1/2} D^{-1/2} U_1^T (H\tilde{a} + h) = H^{-1} h. \quad (4.23)$$

This equation also shows, that the solution vector can be expressed as a linear combination of the eigenvectors u_j (these are the columns of the matrix U_1).

The statistical independence of the components of \hat{a}_1 allows to test the statistical significance of every component independently. If the true value of a certain component is zero (or small compared to 1), the measured value $(\hat{a}_1)_j$ will follow a standard gaussian distribution, and $(\hat{a}_1)_j^2$ will follow a χ^2_1 distribution. Using a confidence level of 95 %, one can consider the j -th component to be compatible with zero, if $(\hat{a}_1)_j^2 \leq 3.84$. If all values $(\hat{a}_1)_j$ with $j > m_0$ are compatible with zero, they can be ignored, and the result can be expressed as a linear combination of the first m_0 eigenvectors. In fact it turns out that those insignificant components are the ones which cause the fluctuations in the full solution. This is seen easily, if the equation (4.20) is rewritten to the form

$$\hat{a} = \sum_{j=1}^m \left(\frac{1}{D_{jj}} \right)^{1/2} (\hat{a}_1)_j u_j. \quad (4.24)$$

Because of the factor $(1/D_{jj})^{1/2}$ (and the unit variance of $(\hat{a}_1)_j$) the insignificant components get a large weight factor in the full solution.

A sharp cut-off in the amplitudes at a certain index m_0 however also introduces some fluctuations in the solution, known as 'Gibbs phenomenon' in the theory of Fourier analysis and reconstruction of periodic functions. A smooth cut-off reducing these oscillations is provided by the regularization method, to be discussed in the next chapter.

4.4 UNFOLDING WITH REGULARIZATION

As explained in the preceeding chapter, unfolding by a straightforward fit without a cut-off will produce a fluctuating result. Mathematically the fluctuations are caused by insignificant components of the solution with their strong oscillations, which get a large weight in the unfolding. The magnitude of the fluctuations can be measured by several quantities; one possible measure is the total curvature, introduced quantitatively in equation (3.02):

$$r(a) = \int [f''(x)]^2 dx. \quad (4.25)$$

One can also consider other measures of the smoothness of the solution, for example based on the square of the first derivative of $f(x)$ [12]. In this paper the discussion is restricted to the measure defined by equation (4.25). This quantity will take on large values for a strongly fluctuating solution, often orders of magnitudes larger than physically motivated expectations. This expectation of a smooth solution, an implicit a priori knowledge, can be used in the unfolding in the following way: a new function $R(a)$ is introduced by adding to the negative log likelihood function the total curvature $r(a)$, weighted by a factor τ :

$$R(a) = S(a) + \frac{1}{2} \tau \cdot r(a) \quad (4.26)$$

(the factor $\frac{1}{2}$ is introduced for later convenience). This method is called regularization method and the factor τ is called regularization parameter. If $f(x)$ is parametrized by a

sum of B-splines of order 4, the choice of equation (4.25) has the particular advantage, that $r(a)$ can be represented by a quadratic expression

$$r(a) = a^T C a \quad (4.27)$$

with a (constant) symmetric, positive semidefinite matrix C ; such a regularization term is easily accounted for in the minimization. Apart from some factor, which can be absorbed in the regularization parameter τ , the matrix C has the form

$$C = \begin{pmatrix} 2 & -3 & 0 & 1 & 0 & 0 & \dots \\ -3 & 8 & -6 & 0 & 1 & 0 & \\ 0 & -6 & 14 & -9 & 0 & 1 & \\ 1 & 0 & -9 & 16 & -9 & 0 & \\ 0 & 1 & 0 & -9 & 16 & -9 & \\ 0 & 0 & 1 & 0 & -9 & 16 & \\ \vdots & & & & & & \ddots \end{pmatrix}$$

for cubic B-splines with equidistant knots.

Obviously, regularization terms can introduce a bias to the solution, which depends on the magnitude of the regularization parameter τ . For $\tau \rightarrow 0$ the effect of the regularization will vanish, and for $\tau \rightarrow \infty$ the result will become a linear function for $r(a)$ defined by equation (4.25). As is shown below, the magnitude of τ can be defined such that the bias will be negligible small compared to statistical errors; in effect, the regularization provides a smooth cut-off of higher order terms in the solution.

Using the transformation (4.20) from a to a_1 , already defined in chapter 4.3, the expression (4.26) to be minimized can be rewritten in the form

$$R(a_1) = -a_1^T D^{-1/2} U_1^T (H\tilde{a} + h) + \frac{1}{2} a_1^T a_1 + \frac{1}{2} \tau \cdot a_1^T D^{-1/2} U_1^T C U_1 D^{-1/2} a_1 \quad (4.28)$$

The regularization term can be written as

$$\frac{1}{2} \tau \cdot a_1^T C_1 a_1 \quad \text{with} \quad C_1 = D^{-1/2} U_1^T C U_1 D^{-1/2}. \quad (4.29)$$

The regularized solution can be calculated using one more transformation of the parameters. The matrix C_1 is transformed to a diagonal matrix S by

$$S = U_2^T C_1 U_2, \quad (4.30)$$

where $U_2^T U_2^T = I_{mm}$; the eigenvalues S_{jj} can be arranged in increasing order $S_{11} \leq S_{22} \leq \dots \leq S_{mm}$. The additional transformation is defined by

$$a_1 = U_2 a' \quad (4.31)$$

and is a pure rotation in parameter space. Note that because of the pure rotation $a_1^T a_1 = a'^T a'$. The rotation yields

$$\frac{1}{2} \tau \cdot a'^T S a' \quad (4.32)$$

for the regularization term and the function to be minimized becomes

$$S(a') = -a'^T U_2^T D^{-1/2} U_1^T (H\tilde{a} + h) + \frac{1}{2} a'^T (I + \tau \cdot S) a'. \quad (4.33)$$

The regularized solution derived from the condition $\nabla S = 0$ is given by

$$\hat{a}' = (I + \tau \cdot S)^{-1} U_2^T D^{-1/2} U_1^T (H\tilde{a} + h), \quad (4.34)$$

whereas the unregularized solution ($\tau = 0$), denoted by a bar, reads

$$\bar{a}' = U_2^T D^{-1/2} U_1^T (H\tilde{a} + h). \quad (4.35)$$

Due to the orthogonality of the rotation matrix U_2 the covariance matrix $V(\bar{a}')$ is still a unit matrix. The result can be transformed back to the coefficients of the basis functions $p_j(x)$ by

$$\bar{a} = U_1 D^{-1/2} U_2 \bar{a}', \quad (4.36)$$

yielding

$$\bar{f}(x) = \sum_{j=1}^m \bar{a}_j p_j(x). \quad (4.37)$$

An equivalent point of view is to consider the transformed set of basis functions $p'_j(x)$, and to write the result in the form

$$\bar{f}(x) = \sum_{j=1}^m \bar{a}'_j p'_j(x), \quad (4.38)$$

where the functions $p'_j(x)$ are linear combinations of the basis functions $p_j(x)$ and are orthogonal and normalized functions¹. In this parametrization the curvature (4.25) is given by

$$\int [\bar{f}''(x)]^2 dx = \sum_{j=1}^m (\bar{a}'_j)^2 S_{jj}. \quad (4.39)$$

A normalized orthogonal function $p'_j(x)$ usually has $(j - 1)$ zeros in the range, and since the eigenvalues S_{jj} are sorted in increasing order, the contribution to the curvature rises rapidly with increasing index j . The analysis of the values of the coefficients \bar{a}' will show, that for $j > m_0$ they are compatible with zero within their statistical errors of 1. With a sharp cut-off at $j = m_0$, the result is expressed by m_0 statistically independent contributions and can be converted to just m_0 data points.

Now the effect of the regularization is considered. Equations (4.34) and (4.35) show, that the coefficients of the regularized solution are

$$\hat{a}'_j = \frac{1}{1 + \tau S_{jj}} \bar{a}'_j. \quad (4.40)$$

¹The linear combinations are determined by the combined effect of the transformations (4.20) and (4.31).

Thus the coefficients of the regularized solution are obtained by the multiplication of the coefficients of the unregularized solution by a factor, which is close to 1 for all indices j with $S_{jj} \ll \tau^{-1}$. Since the values of S_{jj} increase rapidly, the attenuation factor will approach zero for $S_{jj} \gg \tau^{-1}$ after a transition region, where $S_{jj} \approx \tau^{-1}$. Regularization thus means a smooth cut-off, avoiding the already mentioned 'Gibbs phenomenon'. The sum of all factors can be considered as the effective number m_0 of independent contributions to the solution; for a given number m_0 the regularization parameter τ can be defined by

$$m_0 = \sum_{j=1}^m \frac{1}{1 + \tau S_{jj}}. \quad (4.41)$$

The parameter m_0 has to be large enough, such that no significant coefficients are attenuated too much. A lower limit for m_0 can be obtained from statistical tests on the significance of the coefficients \bar{a}'_j . In typical applications the value of m_0 will be chosen just above the lower limit.

Having fixed the value of the regularization parameter τ , the regularized solution can be calculated using equation (4.40). Since the covariance matrix of \bar{a}' is the unit matrix and the regularized solution is $\hat{a}' = (I + \tau \cdot S)^{-1} \bar{a}'$, the covariance matrix of \hat{a}' is

$$V(\hat{a}') = (I + \tau \cdot S)^{-2}. \quad (4.42)$$

The result can then be transformed back by the transformations defined in equations (4.20) and (4.31).

The resulting coefficients have to be converted finally to a set of m_0 data points¹ \hat{f}_k , by integration of $\hat{f}(x)$ over small regions of x according to equation (4.17). The choice of these regions has of course consequences for the correlations between the data points, which should be as small as possible. One method to achieve this is the following. Since the function $p'_{m_0+1}(x)$ has just m_0 zeros, it seems optimal to define the m_0 regions around the zeros, with the $(m_0 - 1)$ x -values, where the function $p'_{m_0+1}(x)$ has extreme values, taken as limits. This has the effect of suppressing the contribution of the term $\hat{a}'_{m_0+1} p'_{m_0+1}(x)$, which is attenuated by a factor of roughly $1/2$, and takes into account the statistical precision and the resolution as a function of x . Each average value is expressed by a linear combination of the coefficients \hat{a}'_j or \hat{a}_j , and error propagation is straightforward. Using the unfolding result $\hat{f}(x)$ as a weighting factor for the MC-events, the quality of the description of the measured distributions can be tested. This test can be extended to measured variables not directly used in the unfolding fit.

Numerical example of unfolding with regularization. In this example the Monte Carlo technique is used for the simulation of a measurement with limited acceptance and limited resolution. The measurement of a variable x with $0 \leq x \leq 2$ is simulated with the following properties of the measurement: The acceptance probability is assumed to be

$$P_{acc}(x) = 1 - \frac{1}{2}(x - 1)^2; \quad (4.43)$$

¹One could of course choose a larger number of data points, however the rank of the covariance matrix of this set of data points will be m_0 .

the true values x are transformed by the function

$$y_{tr} = x - 0.2 \frac{x^2}{4} \quad (4.44)$$

to a variable y_{tr} , which is then assumed to be measured with a gaussian resolution function with a standard deviation $\sigma = 0.1$, resulting in the measured variable y . The assumed acceptance, the transformation function and the resolution function is shown in Figure 7 a, the response of the simulated detector to δ -function signals at $x = 0.5$, $x = 1.0$ and $x = 1.5$ is shown in Figure 7 b.

The assumed true function is

$$f(x) = \sum_{k=1}^3 b_k \frac{g_k^2}{(x - x_k)^2 + g_k^2} \quad (4.45)$$

in the region $0 \leq x \leq 2$; the parameters used in the simulation are given in table 1.

k	b_k	x_k	g_k
1	1.0	0.4	2.0
2	10.0	0.8	0.2
3	5.0	1.5	0.2

Table 1. Parameters of true function.

A sample of 5000 random x -values is generated according to the function $f(x)$. A histogram of the sample is shown in Figure 8 a together with the function $f(x)$. After acceptance, transformation and smearing according to the assumptions made above 4475 y -values remain; a histogram of this sample, representing the result of the measurement, is shown together with the original function $f(x)$ in Figure 8 b.

The transformation restricts y_{tr} to $0 \leq y_{tr} \leq 1.8$, with the additional effect, that the two peaks become slightly narrower. The resolution function then broadens the peaks, filling up the valley between the peaks.

Now the unfolding method with regularization is applied to the data. The number of spline functions used in the discretization is $m = 22$. In table 2 several parameters of the unfolding method are shown, including the coefficients of the transformed solution. These values are also shown in Figure 9 a as bars. As is seen, about half of the coefficients are small and compatible with zero. The eigenvalues S_{jj} of the curvature matrix C , also given in table 2, increase by many orders of magnitude. A few of the orthogonal functions $p'_j(x)$ are shown in Figure 10. The amplitudes of the functions, each representing one standard deviation statistical error, increase with increasing index j . The regularization

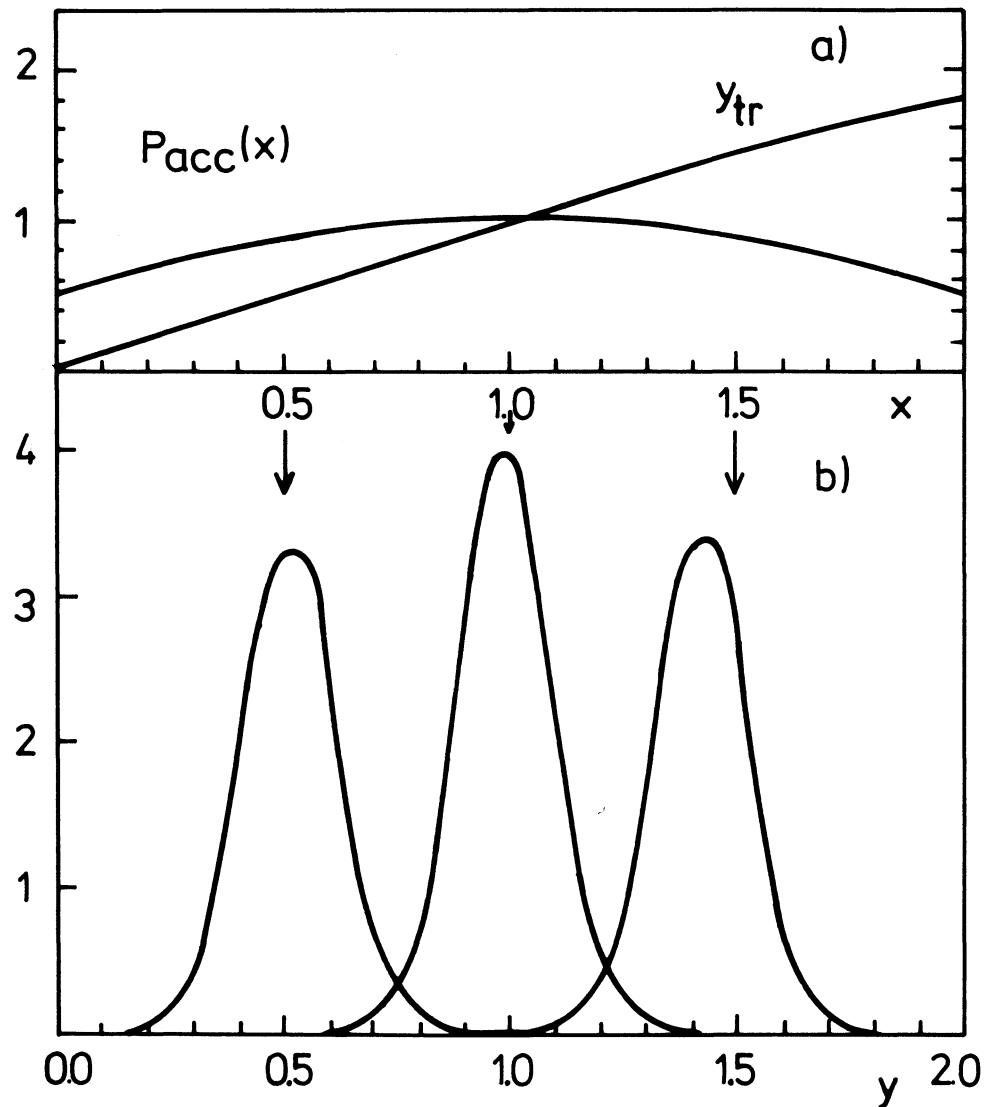


Figure 7. Acceptance, transformation and resolution, assumed in the simulation (a) and response to δ -functions at the values, indicated by arrows (b).

parameter τ is determined in this example for $m_0 = 12$, the result obtained from equation (4.41) is $\tau = 0.4287 \cdot 10^{-3}$. The coefficients of the regularized solution, obtained by multiplying the coefficients \bar{a}'_j with the factors, as given by equation (4.40), and the factors themselves are given in the last two columns of table 2. The coefficients of the regularized solution are also shown in Figure 9 a. Figure 9 b shows the attenuation factor as a function of the index j .

Finally the set of coefficients is converted to data points, representing average values

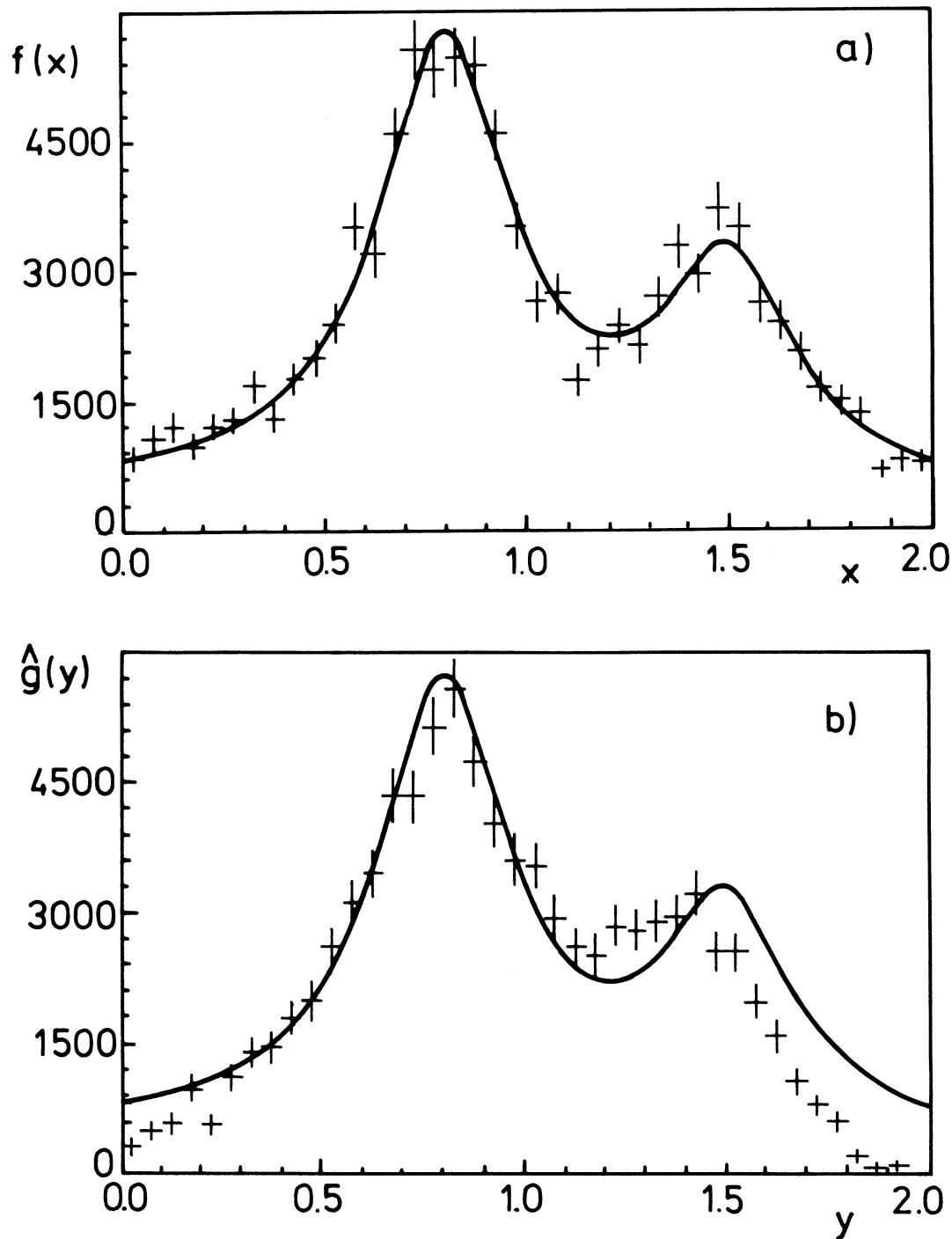


Figure 8. Histogram of the generated data (a) before and (b) after simulation of acceptance, transformation and resolution. The original function is shown as curve.

j	S_{jj}	\bar{a}'_j	\hat{a}'_j	factor
1	0.02	64.62	64.62	1.0000
2	0.09	1.70	1.70	1.0000
3	0.28	7.51	7.51	0.9999
4	0.79	9.86	9.85	0.9997
5	1.97	1.27	1.27	0.9992
6	4.86	11.23	11.21	0.9979
7	12.32	0.18	0.18	0.9947
8	29.61	3.72	3.67	0.9875
9	72.43	0.68	0.66	0.9699
10	183.90	0.04	0.04	0.9269
11	507.00	0.63	0.52	0.8214
12	1409.32	1.38	0.86	0.6233
13	$3.54 \cdot 10^3$	1.81	0.72	0.3974
14	$11.86 \cdot 10^3$	0.03	0.01	0.1644
15	$41.45 \cdot 10^3$	1.38	0.07	0.0533
16	$98.04 \cdot 10^3$	0.05	0.00	0.0232
17	$129.89 \cdot 10^3$	1.27	0.02	0.0176
18	$209.10 \cdot 10^3$	0.41	0.00	0.0110
19	$334.44 \cdot 10^3$	0.52	0.00	0.0069
20	$423.96 \cdot 10^3$	1.59	0.01	0.0055
21	$10.51 \cdot 10^6$	0.80	0.00	0.0002
22	$38.64 \cdot 10^6$	0.30	0.00	0.0001

Table 2. Parameters of the unfolding solution.

for the unfolded $f(x)$ in small x -regions. As explained above, the limits of these regions are determined by the positions of the extrema of the function $p'_{m_0+1}(x)$, in this case of the function $p'_{13}(x)$, which is shown in Figure 10 b, with the positions of the extrema indicated. As can be seen from Figure 10 b, due to the definition of the region limits the contribution of the function $p'_{13}(x)$ itself will be very small for the average values. The final result is shown in Figure 11 together with the original function $f(x)$. The unfolded data are within errors compatible with the function, in particular the valley between the peaks, hardly visible in the 'measured' data (Figure 8 b), is reproduced. Of course the statistical errors are much larger than in the histogram of the generated data in Figure 8 a. They illustrate the loss in statistical accuracy, that occurs due to a measurement with finite resolution.

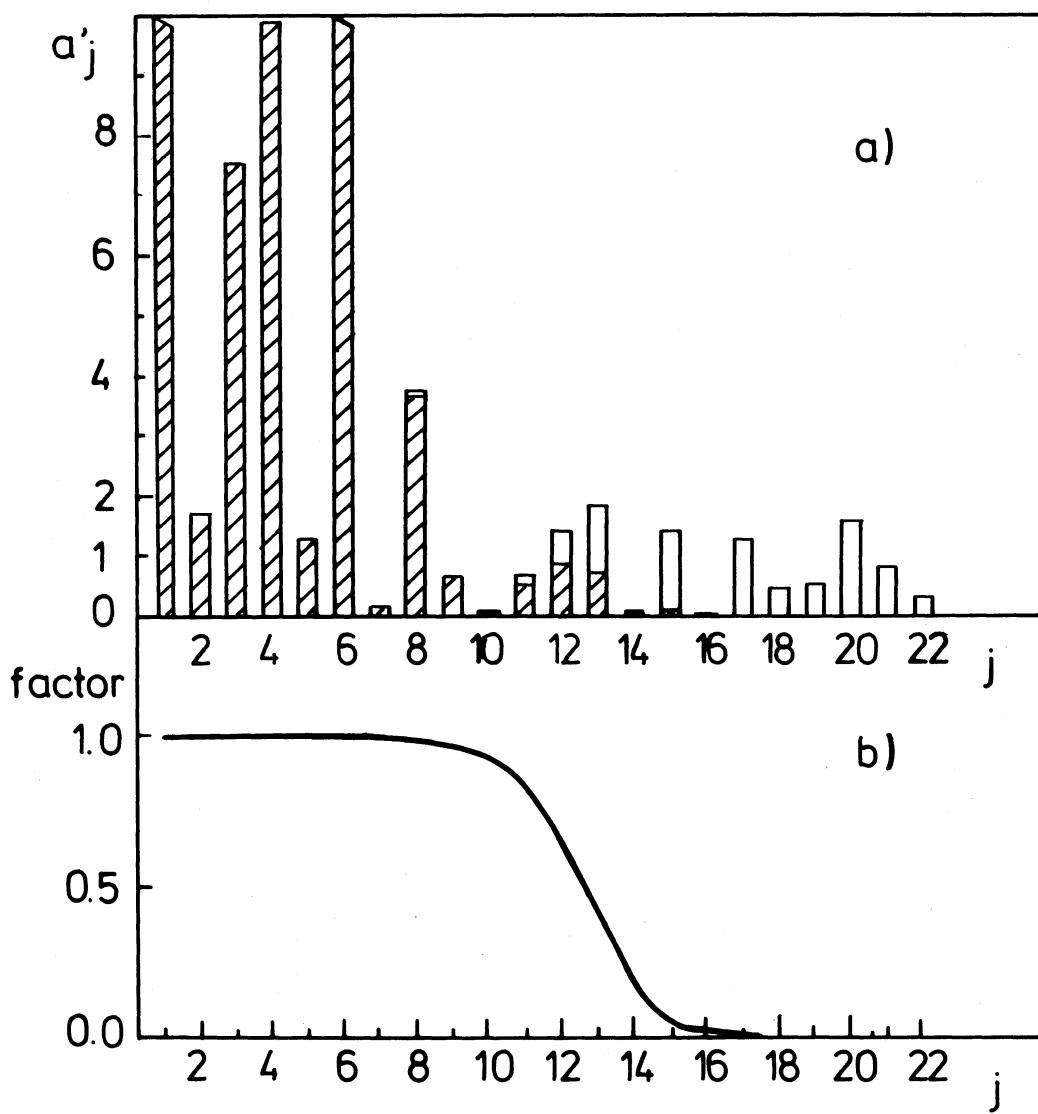


Figure 9. Plot of the coefficients of the normalized orthogonal functions of the solution (a); the dashed bars represent the values after the regularization. The attenuation factor is shown as a curve (b).

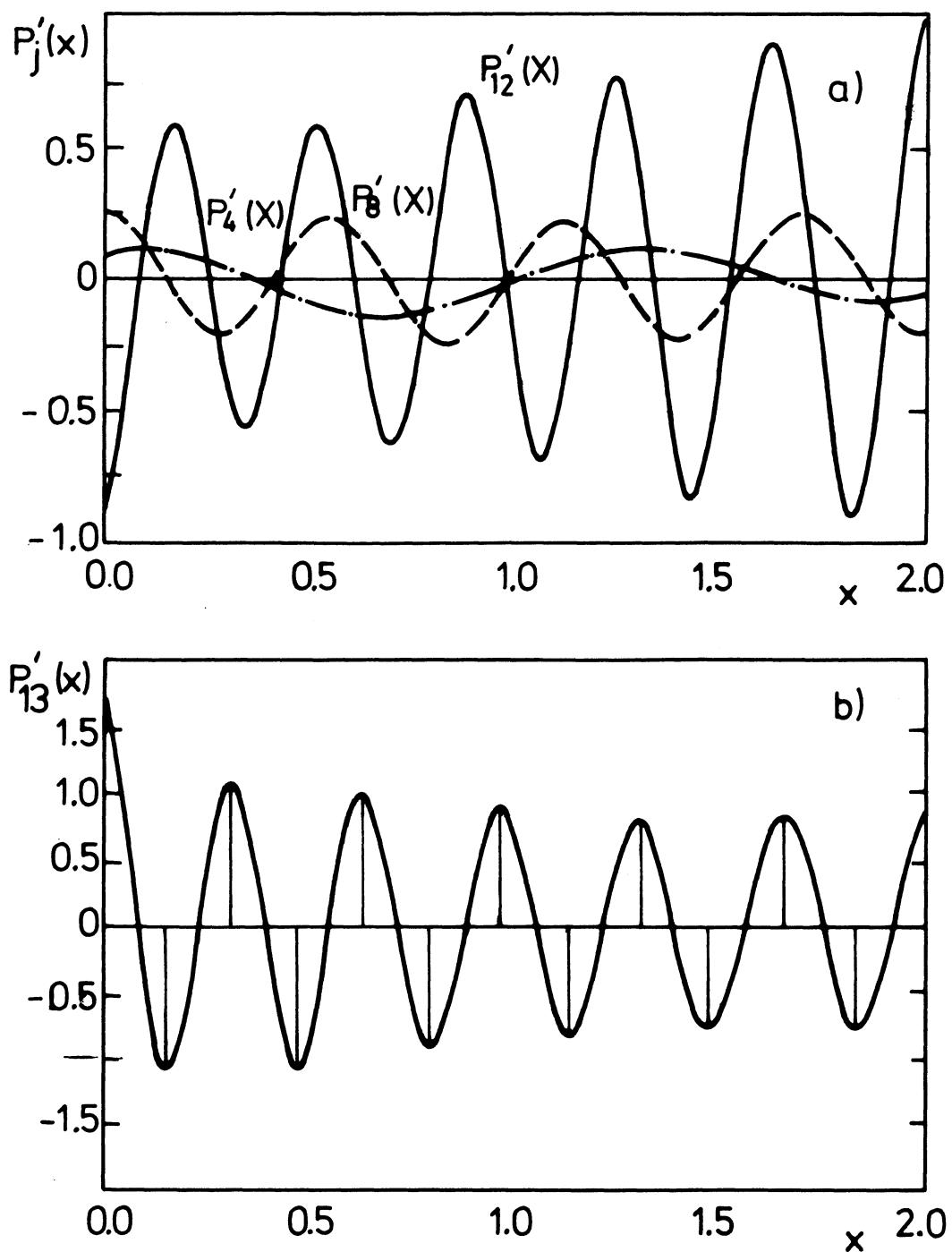


Figure 10. Examples for the normalized orthogonal functions, which are used to represent the solution.

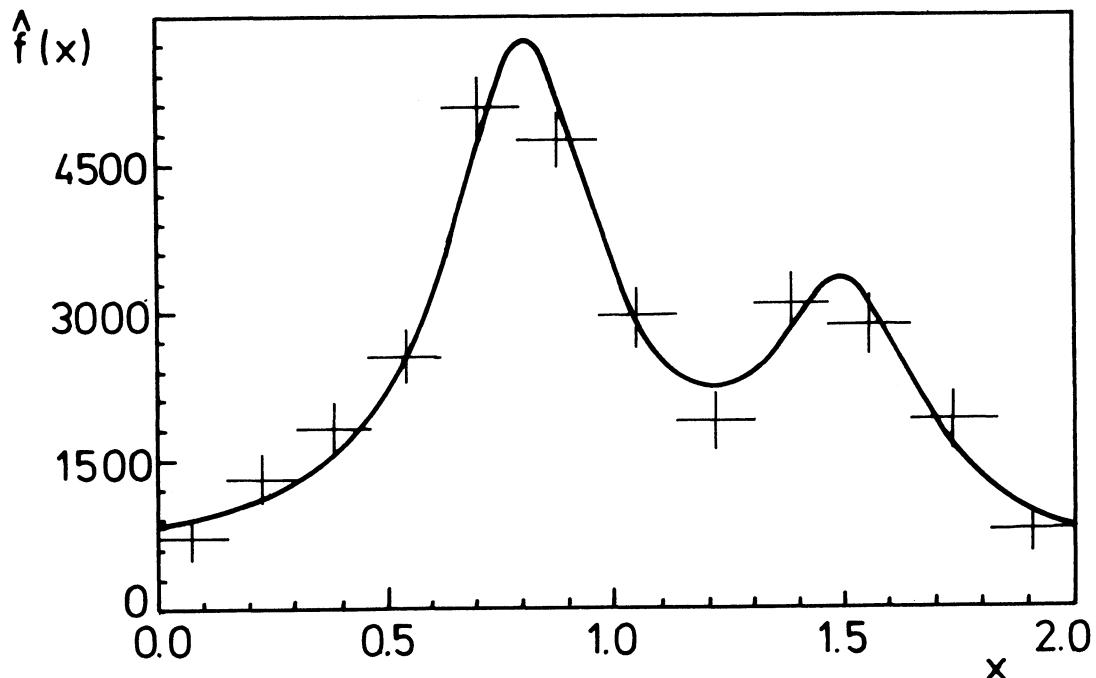


Figure 11. Result of unfolding with regularization, shown as data points together with the original function. The horizontal bar gives the range, over which the data points represents the average.

In conclusion, the regularization, which is the essential component of the unfolding method derived here, is shown to result in a smooth cut-off of insignificant higher order terms in the expansion of the unfolding solution in terms of orthogonal functions. The inclusion of these higher order terms would produce a spurious oscillatory component of the solution. Standard statistical tests are used for the determination of the regularization weight; they ensure that a possible bias introduced by the regularization is small compared to the statistical errors. The method also contains a prescription for the definition of a set $\{\hat{f}_k\}$ of average function values of the unfolded solution, which are only weakly correlated. It should be stressed, that unfolding cannot modify the statistical accuracy of an experiment, which is always reduced by the effect of limited resolution.

ACKNOWLEDGEMENT

The method discussed in chapter 4.4 of this paper has been applied to several high-energy physics experiments in an earlier and in the present version. For discussions and suggestions during the development I have to thank many colleagues, in particular J. V. Allaby, L. Criegee, J. Dainton, U. Eckardt, R. Orr, K. H. Ranitzsch, I. Scillicorn and K. Winter. I want to thank L. Criegee and I. Scillicorn for a critical reading of the manuscript.

REFERENCES

1. J. H. Friedman, *Data analysis techniques for high energy particle physics*, in "Proceedings of the 1974 CERN School of Computing, Godoysund, Norway", CERN 74-23, 1974.
2. J. Carr, *Analysis of deep inelastic muon and electron scattering experiments*, in "Formulae and Methods in Experimental Data Evaluation with Special Emphasis on High Energy Physics", Vol. 2, European Physical Society, CERN, 1984.
3. I. P. Nedelkov, *Improper problems in computational physics*, Comp. Phys. Comm. **4** (1972), 157-164.
4. S. Christiansen, *Integral equations: An outline*, in "Formulae and Methods in Experimental Data Evaluation with Special Emphasis on High Energy Physics", Vol. 3, European Physical Society, CERN, 1984.
5. B. W. Rust and W. R. Burrus, "Mathematical Programming and the Numerical Solution of Linear Equations", American Elsevier Publishing Company, Inc., New York, 1972.
6. M. Jonker et al. (CHARM Collaboration), *Experimental study of differential cross sections $d\sigma/dy$ in neutral current neutrino and antineutrino interactions*, Physics Letters **102 B** (1981), 62-72.
7. Ch. Berger et al. (PLUTO Collaboration), *Measurement of the photon structure function $F_2^{\gamma}(x, Q^2)$* , Physics Letters **142 B** (1984), 111-118.
8. Ch. Berger et al. (PLUTO Collaboration), *Measurement of deep inelastic electron scattering off virtual photons*, Physics Letters **142 B** (1984), 119-124.
9. D. Drijard, *Design of experiments*, in "Proceedings of the 1978 CERN School of Computing, Jadwisin, Poland", CERN 78-13, 1978.
10. D. L. Phillips, *A technique for the numerical solution of certain integral equations of the first kind*, J. Assoc. Comput. Mach. **9** (1962), 84-97.
11. A. N. Tikhonov and V. Ya. Arsenin, "Metody resheniya nekorrektnyh zadach", (Methods for Solution of ill-posed Problems), Nauka, Moscow, 1979.
12. J. Routti and J. V. Sandberg, *General purpose unfolding program LOUHI78 with linear and nonlinear regularization*, Comp. Phys. Comm. **21** (1980), 119-144.
13. S. W. Provencher, *A constrained regularization method for inverting data represented by a linear algebraic or integral equation*, Comp. Phys. Comm. **27** (1982), 213-227.
14. V. P. Zhigunov, *Improvement of resolution function as an inverse problem*, Nucl. Instrum. and Methods **216** (1983), 183-190.
15. M. Jonker et al. (CHARM Collaboration), *Experimental study of x -distributions in semileptonic neutral current neutrino interactions*, Physics Letters **128 B** (1983), 117-123.

16. W. T. Eadie, D. Drijard, F. E. James, M. Roos and B. Sadoulet, "Statistical Methods in Experimental Physics", North-Holland Publishing Company, Amsterdam, London, 1971.
17. V. Blobel, *Least squares methods*, in "Formulae and Methods in Experimental Data Evaluation with Special Emphasis on High Energy Physics", Vol. 3, European Physical Society, CERN, 1984.
18. J. Stoer and R. Bulirsch, "Introduction to Numerical Analysis", Springer-Verlag, New York, Heidelberg, Berlin, 1980.
19. H. Wind, *Interpolation and function representation*, in "Formulae and Methods in Experimental Data Evaluation with Special Emphasis on High Energy Physics", Vol. 3, European Physical Society, CERN, 1984.
20. C. de Boor, "A Practical Guide to Splines", Springer-Verlag, New York, Heidelberg, Berlin, 1978.
21. R. Courant und D. Hilbert, "Methoden der Mathematischen Physik I", Springer-Verlag, Berlin, 1924.
22. C. Lanczos, "Applied Analysis", Prentice-Hall, Inc., Englewood Cliffs, N.J., 1956.
23. E. L. Kosarev and E. Pantos, *Optimal smoothing of noisy data by fast fourier transform*, Sci. Instrum. **16** (1983), 537-543.

MONTE CARLO TECHNIQUES

J. Salicio

Junta de Energía Nuclear, Madrid/Spain

and

Deutsches Elektronen-Synchrotron DESY, Hamburg/Fed. Rep. of Germany

ABSTRACT

The course of "Monte Carlo Techniques" will try to give a general overview of how to build up a method based on a given theory, allowing you to compare the outcome of an experiment with that theory.

Concepts related with the construction of the method, such as, random variables, distributions of random variables, generation of random variables, random-based numerical methods, will be introduced in this course.

Examples of some of the current theories in High Energy Physics describing the e^+e^- annihilation processes (QED, Electro-Weak, QCD) will also be briefly introduced.

A second step in the employment of this method is related to the detector. The interactions that a particle could have along its way, through the detector as well as the response of the different materials which compound the detector will be quoted in this course.

An example of detector at LEP era, in which these techniques are being applied, will close the course.

1. INTRODUCTION

Many times an experimental physicist has the problem of how to compare the outcome of an experiment with the prediction of a theory. Some times the features of the experiment and/or the uncomplexity of the theory allows a direct comparison of the analytical expression of the theory with the experimental distributions.

The current High Energy Physics panorama, although from the formal point of view looks simpler (less theories and fewer number of free parameters to describe them), becomes more and more complex in its implementation. Larger analytical representations force to consider series expansions to some degree of accuracy. The experimental apparatus also raise in complexity since the aim is to collect as much information as possible and analyze it quickly.

The detectors, made out of many components with different purposes, are not as perfect as one would like. Usually materials integrating a detector have fluctuations in their behavior, for instance, with respect to the "passage" of a particle through them. In addition, most of the detectors do not have a complete acceptance, i.e. not all the possible particles produced in a process will go through active parts of the detector.

This situation leads to the search of some tools allowing us to reproduce similar effects from the theory to those that a real experiment could produce.

The randomness with what the real data are produced, to finally fill up the distributions predicted by theories, provides the foundations of the tool we look for. Indeed, the solution comes from the possibility of generating random numbers, that, under some constraints, shall allow us to reproduce the predictions of the theories.

Normally in High Energy Physics, the observables of an experiment are the particles that have been produced as result of a collision of two particles (for instance, e^+e^-

collision, $\pi^- p$ collision, etc). What we call "an event" is the set of particles defined by their fourmomenta produced as result of the collision. A sample of such events is what we compare with a collection of "Monte Carlo events" which were generated according with the theory to test.

When the outcome of the experiment is not perfect, because the detector, i.e. when the detector does not see all the events produced, or does not see all particles of every event, or the measurement of the momentum and energy of the particles is made with some error, etc, the comparison with the predictions of the theory is harder, but still possible by simulating the behavior of particles created from the theory, when passing through different components of the detector. Since detectors are made out of materials, there are particle-matter interactions which should be simulated.

This course will consist of two main items:

- 1) The foundations to make any simulation in which the generation of random numbers and operations with them are necessary.
- 2) The simulation of the physical process (Theory) and the detection process (Detector) making use of known properties of matter.

2. GROUNDs

I do not want to give an exhaustive review of random processes but a general feeling and few tools we shall need to apply to the problem of simulation in High Energy Physics (H.E.P.).

In H.E.P., relativistic and quantic effects become essential and this last introduces a basic uncertainty in the observables. In other words, the problem of given an initial state, have the final state, perfectly determined has no deterministic solution and has to be answered in terms of probabilities: "probability of finding a given final state configuration. Here comes out the stochastic nature of real physical processes in which the studies are made based on probabilities. [1] As an example consider the H.E.P. process $e^+ e^- \rightarrow \mu^+ \mu^-$ (studied in a later chapter) and suppose that only the diagram with γ exchanged in the s channel contributes to it. If we make an experiment searching for final states with only μ^+ and μ^- and we consider the angular distribution of the μ^- in the center of mass frame of the colliding $e^+ e^-$, it comes out an $A + B \cos^2 \theta$ distribution. This is one of the typical random distributions that we can find in H.E.P. It says what is the probability of finding a μ^- produced in a given angular range. If you repeat several times the same experiment, you will end up with similar distributions.

2.1 Random variables, samples

The variables taking values which can not be predicted in advance are called random variables. [2] The $\cos \theta$ variable in the previous example was a random variable. The predictions of theories are normally given in terms of distributions of random variables or equivalently in terms of probabilities of finding given values in given intervals. In that sense we can define the probability density function $g(x)$ as the probability of finding x' in the range x and $x + dx$

$$g(x)dx = P(x'; x < x' < x + dx) \quad (2.1)$$

A function y of a random variable x , $y = y(x)$ is a random variable if it is continuous and derivable. That is, the random nature of variables does not change with transformations.

This feature of random variables will be useful at the time in which we shall try to reproduce distributions of a given theory, since we shall use mechanisms to generate special kind of random distributions as are the uniform, Gaussian, or Poisson distributions and we shall need to transform them to others.

Let's call sample $S_k(g)$ to a distribution of values of a random variable x distributed according to a probability density function $g(x)$. In practice, a sample $S_k(g)$ may be thought as the observed result of an experiment k

2.2 Cumulative distribution function

Let a distribution of random variables x be described by $g(x)$ and defined in the interval $[a, b]$, then we define the cumulative distribution function of x as

$$G(x) = \int_a^x g(x') dx' = P(a \leq x' \leq x) \quad (2.2)$$

That is the probability of finding x' in the range $[a, x]$. This function (also called distribution function of x) has the property of being monotonically non-decreasing, $|3|$ taking values from "0" to "1". The last assertion is true since we have defined $g(x)$ as a probability density function, otherwise it can always be normalized inside the definition range.

This function $G(x)$ has the property to transform non-uniform variations in the x variable into uniform variations in the $G(x)$ variable

$$d G(x) = g(x) dx \quad (2.3)$$

Consider the distribution of values of the x variable shown in fig. 1, given by

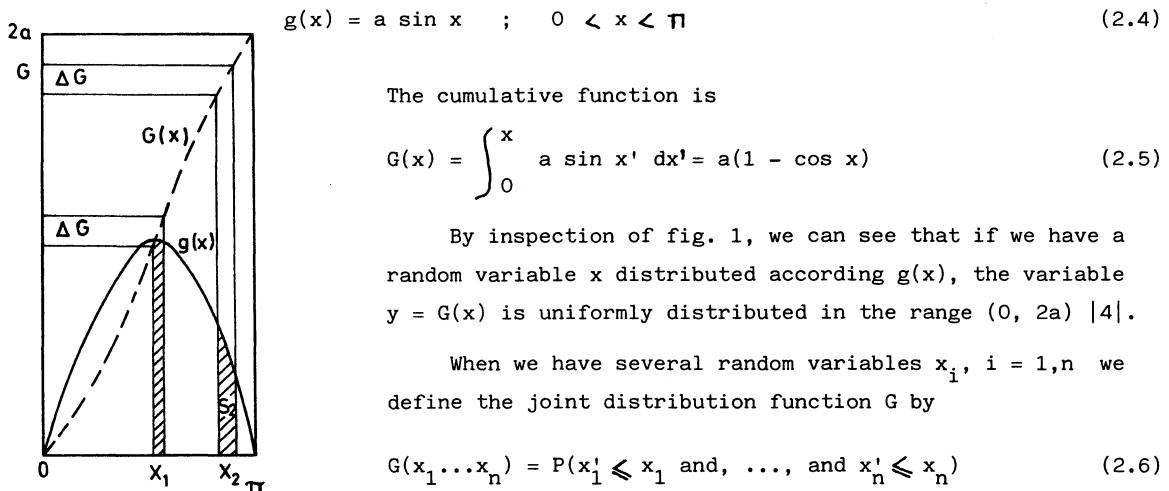


Fig.1 Representation of the probability density function $g(x)$ and its cumulative distribution function $G(x)$.

The random variables $x_1 \dots x_n$ are independent when

$$P(x'_1 \leq x_1 \text{ and } \dots \text{ and } x'_n \leq x_n) = \prod_{i=1}^n P(x'_i \leq x_i) \quad (2.7)$$

If $G(x_1 \dots x_n)$ is differentiable in each variable, the joint density function is given by

$$g(x_1 \dots x_n) = \frac{\partial^n G(x_1 \dots x_n)}{\partial x_1 \dots \partial x_n} \quad (2.8)$$

2.3 Generation of random variables

Here we give some methods to generate random numbers. In fact what we shall see are not properly random numbers, in the sense that we formulate some algorithm to generate sequences of numbers which have properties very close to those of random numbers. Usually these numbers are called pseudo-random. Obviously, a random number can not be generated through any kind of algorithm, since one will be able from a given number to predict the next one*). Nevertheless, for the purpose of simulation, pseudo-random numbers satisfy the necessary requirements.

2.3.1 Random variables uniformly distributed

A uniform distribution of random variables x , is that having a constant probability density function

$$g(x) = \text{constant} \quad (2.9)$$

There are several techniques to produce uniform distributions of random variables in computers. The most commonly used are the so called "congruential procedures", since they are very fast in generating and they can be easily analyzed. The attractiveness of the congruential method described below [5] lies in their general susceptibility to theoretical analysis and in the ability of replace multiplications and divisions by shift and add instructions. These, called multiplicative congruential procedures, take the form:

$$x_{n+1} = \alpha x_n + \beta \quad (2.10)$$

where x_0 , α , β and m are parameters to be defined. When $\beta = 0$ one speaks about a "pure multiplicative congruential" procedure. Many suggestions have been made for the proper choice of the parameters [6], but for simplicity we choose $\alpha = 2^{(P/2)+1} + 3$, $\beta = 0$ and $m = 2^P$ for binary computers with word length of P -bits. In a computer such as the VAX 780 or the IBM 3081, with a word length of 32 bits, the choice would be $\alpha = 2^{17} + 3$, $\beta = 0$ and $m = 2^{32}$.

For the analysis of the properties of the sequences of numbers generated with algorithm above, I remit the reader to the specialized literature. For us, it should be enough to know that it satisfies the minimum requirements of big periodicity and very small correlation between succeeding terms. In practice, the sequence $\{x_n / m\}$ is taken to be uniformly distributed over the range $(0, 1)$.

CERN program libraries [7] have programs to generate random numbers. For instance the

*) Von Neumann: "Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin!"

routine RNDM2 (V107) is a uniform pseudo-random generator based in the combination of a multiplicative congruential procedure and a shift register generator. The performance in the IBM is very good and has a period of about 5×10^{18} numbers.

2.3.2 Non-uniform distributions

Once we are able to generate random uniform distributions a wide variety of non-uniform distributions can be generated. The method will be "direct" if the cumulative distribution function, $y = G(x)$, of a given distribution has inverse

$$x = G^{-1}(y) \quad (2.11)$$

In that case, through a random uniform generation of the variable y , we get the desired distribution.

Consider the example given in paragraph 2.2. The cumulative distribution function was $G(x) = a(1 - \cos x)$. This function has inverse

$$x = G^{-1}(y) = \arccos(1 - \frac{y}{a}); \quad 0 < y < 2a \quad (2.12)$$

Then if y is a random variable uniformly distributed over the range $[0, 2a]$, the variable $x = G^{-1}(y)$ should be distributed as: $a \sin x$.

Unfortunately, not all the non-uniform distributions we want to generate, have a cumulative distribution function easily invertible.

A general method of obtaining a random sequence, is the "acceptance-rejection" method, that in general is not efficient and needs two different sequences of random numbers uniformly distributed:

Let $\{x_o\}$ and $\{y_o\}$ be two independent sequences of random numbers uniformly distributed, then the sequence

$$\{x ; g(x) \geq y, \quad x \in \{x_o\}, \quad y \in \{y_o\}\} \quad (2.13)$$

is $g(x)$ -random distributed. Let us suppose that $g(x)$ takes values in $[a, b]$, then, the efficiency of the method can be defined as the ratio

$$\epsilon = \frac{\int_a^b g(x)dx}{(b-a)(g_{\max} - g_{\min})} \quad (2.14)$$

where g_{\max} and g_{\min} are the maximum and minimum values of $g(x)$ in its definition range.

Another method that can be very efficient under certain conditions is the "Importance sample" method. If $g(x)$ is the distribution to be generated, the method consists in finding a function $f(x)$ approximating $g(x)$, with the cumulative distribution function $F(x)$ having inverse

$$x = F^{-1}(y) \quad (2.15)$$

In that case we can take following procedure to generate $g(x)$

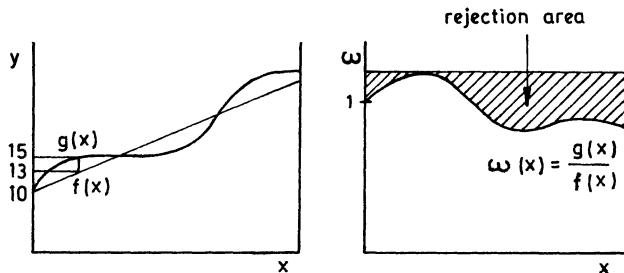
1^o) take y uniformly generated over the range $[F_{\min}, F_{\max}]$ and compute $x = F^{-1}(y)$. Then x is $f(x)$ -random distributed.

2^o) take ω uniform over the range $[0, (g/f)_{\max}]$

3^o) Reject x if $\omega > g(x)/f(x)$, otherwise take it.

The sequence of remaining numbers $\{x\}$ after repeating many times the above steps is $g(x)$ -random distributed.

The method is more efficient when the ratio $\omega(x) = g(x)/f(x)$ is nearly constant or has a slow variation with x . Care must be taken in regions where $g(x)$ is very small or "0" when compared with $f(x)$.



Some other methods are still available to generate non-uniform random distributions, as for instance stratified sampling. It will be quoted in next paragraphs when talking on Monte Carlo integration.

Fig.2. Scheme of the importance sample procedure of generating random distributions

2.4 Numerical methods random-based. Monte Carlo integration

There are a few methods based in random numbers that allow computation of integrals, derivatives, etc. Among these, we are mainly interested in Monte Carlo integration, since it is needed in the computation of quantities, such as cross sections. An excellent review is given in ref [2].

2.4.1 Expectation and variance of a function

The expectation of a function $f(x)$ is defined as the average of the function:

$$E(f) = \int f(x) g(x) dx \quad (2.16)$$

where $g(x)$ is the probability density function of variable x . In particular, if $f(x) = x$, its expectation is

$$E(x) = \int x g(x) dx \quad (2.17)$$

The variance of a function, $f(x)$, is defined as the quadratic deviation of f from its expectation

$$V(f) = E(f - E(f))^2 = \int [f(x) - E(f)]^2 g(x) dx \quad (2.18)$$

In the case of a function of several variables, its expectation is defined as

$$E(f) = \int \dots \int f(x_1 \dots x_n) g(x_1 \dots x_n) dx_1 \dots dx_n \quad (2.19)$$

where $g(x_1 \dots x_n)$ is the joint probability density function describing the distribution of

variables $x_1 \dots x_n$.

2.4.2 Law of large numbers

The law of large numbers says that if f is a function of the random variable x and $\{x\}$ is uniformly distributed in the interval $[a, b]$, and $V(x)$ is finite, then

$$\lim_{n \rightarrow \infty} \frac{1}{n} \sum_{i=1}^n f(x_i) = E(f) = \frac{1}{b-a} \int_a^b f(x) dx \quad (2.20)$$

This law constitutes the fundation of the Monte Carlo integration method. The integral $\int_a^b f(x) dx$ can be aproched with the sum of n random numbers $f(x_i)$. The method is powerful in spite it has a slow convergence to the value of the integral.

2.4.3 Central limit theorem

The sum of a large number of independent random variables is asymptotically normal-distributed, no matter how the individual random variables are distributed.

A normal distribution is characterized by giving the mean value, μ , and the variance, σ^2

$$N(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (2.21)$$

Since each term in a Monte Carlo integration is a random variable from the same distribution, the estimate

$$\bar{I} = \frac{1}{n} \sum_{i=1}^n f(x_i) \quad (2.22)$$

will be asymptotically normal-distributed, with mean equal to the true value of the integral and variance

$$V(\bar{I}) = \frac{1}{n} V(f) \quad (2.23)$$

since the standard error on \bar{I} is the square root of $V(\bar{I})$, this gives the well known result that Monte Carlo estimates converge as $1 / \sqrt{n}$

$$S^2(I) = \frac{1}{n} V(f) \quad (2.24)$$

2.4.4 Variance reducing techniques

In this paragraph I give an overview of the most current techniques to improve the convergence of the Monte Carlo integration method. In the previous paragraph we have seen that the uncertainty in the integral, $S(\bar{I})$, variates as $\sqrt{V(f)} / n$. The uncertainty can be improved, either increasing the number of trials, n , or decreasing the value of the variance $V(f)$. Some of the methods used to decrease the variance are briefly described below.

Stratified sampling

This technique consists in integrate the function in several subregions $\{j\}$ of the total integration region and compute partial sums in each subregion $\{j\}$, where n_j points

are taken

$$\bar{I} = \sum_j \frac{v_j}{n_j} \sum_{i=1}^{n_j} f(x_i) \quad (2.25)$$

The variance is formed through sums over each region weighted proportionally to the volume of region $\{j\}$ and inversely to n_j

$$V(\bar{I}) = \sum_j \frac{v_j}{n_j} \left| \int_{\{j\}} f^2(x) dx - \sum_j \frac{1}{n_j} \left| \int_{\{j\}} f(x) dx \right|^2 \right| \quad (2.26)$$

A particular case, is the uniform stratification, in which the integration region is divided into equal volumes and the chosen number of points is equal in each.

Importance sample

This method relies in the possibility of approximating a function $f(x)$ with another function $g(x)$, easy enough as to be invertible or to have a generator reproducing it. Mathematically, if f is the function we want to integrate, the method corresponds to a change of variable

$$\begin{aligned} x &\rightarrow g(x) \\ f(x)dx &\rightarrow f(x) \frac{dG(x)}{g(x)} \end{aligned} \quad (2.27)$$

such that points, instead being chosen uniformly distributed in x are chosen from a $g(x)$ -distributed sequence. Then f is inversely weighted by $g(x) = dG(x) / dx$. The variance is now $V(f/g)$ which can be small if g is close to f .

The method has problems when $g(x)$ is very small or "0" when compared with $f(x)$.

Control variates

This method similar to the previous one, looks also for some function $g(x)$ that approximates $f(x)$, but this time the function is subtracted rather than divided

$$\int f(x) dx = \int (f(x) - g(x)) dx + \int g(x) dx \quad (2.28)$$

If $g(x)$ is known over the whole integration range, the only uncertainty comes from the integral of $f-g$ which will have a smaller variance. Moreover, zeros of g will not produce singularities in $f-g$. Another advantage is that g needs not to be analytically inverted.

In general, the two last methods need a good knowledge of the function $f(x)$ in all the integration range.

3. APPLICATIONS IN HIGH ENERGY PHYSICS

We have seen how to generate distributions of numbers according to some given function. Now, we can go a step further and describe how to produce "events" according to a theory and later, how the constituents of the events (particles), can be detected.

3.1 Simulation of theories

We describe, now, some of the theories we deal with from the point of view of our interest, i.e. preparing the way for the simulation. Some of the problems related with the lack of knowledge or with the extremely complicated representation of the theory, will be quoted. Problems such as radiative corrections, or fragmentation and hadronization in QCD, become very important as the available energies at PETRA or PEP in e^+e^- or at the SPS in $p\bar{p}$, introduce sizeable effects.

3.1.1 QED and Electroweak

Let us look to the process

$$e^+e^- \rightarrow \mu^+\mu^- \quad (3.1)$$

that can be mediated by γ or by the neutral vector boson Z_0 . The lowest order cross section is given by

$$\frac{d\sigma_0}{d\ln\mu} = \frac{\alpha^2}{4s} [w_1(s) (1 + \cos^2\theta) + w_2(s) \cos\theta] \quad (3.2)$$

where s is the center of mass energy squared, θ is the angle between e^+ and μ^+ and w_1, w_2 are parameters depending on s , the mass M_Z and width Γ_Z of the neutral boson and the vector and axial coupling constants C_V, C_A . Only diagrams in fig. 3 contribute to the above cross section. Higher order corrections, as bremsstrahlung and virtual corrections, must be included when \sqrt{s} is close to the Z_0 mass. A treatment of an almost α^3 order corrections to the μ -pair production and in general fermion-pair production can be found in Ref [8]. In it, the interference between diagrams in figures 3 and 4 and the whole contribution from bremsstrahlung diagrams in fig. 5, are included.

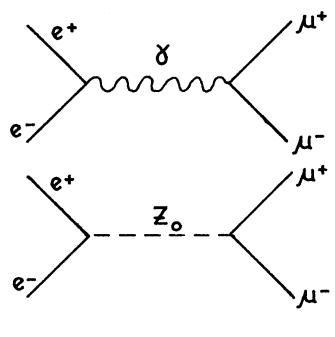


Fig.3 Diagrams contributing to the lowest order cross section for process $e^+e^- \rightarrow \mu^+\mu^-$

Here I only quote the procedure to generate events without entering in details.

To generate events, two regions should be considered: The soft bremsstrahlung region and the hard bremsstrahlung region. Infrared divergences in the soft region are cancelled out with the vertex corrections and box diagrams. (Fig 4a, c, d).

The separation between soft and hard regions is made through the parameter K_0 , which is the photon momentum, choosed to be 1% of the beam energy, $E_{beam} = \sqrt{s}/2$.

Since the cross section can be computed in both regions

$$\sigma_{Total}^{Soft} ; K \leq K_0 \quad (\text{by numerical integration})$$

$$\sigma_{Total}^{Hard} ; K > K_0 \quad (\text{by analytical integration})$$

we can choose at random in what region to generate an event. That can be done by generating a random variable x' uniformly distributed in $(0, 1)$. Then if

$$0 < x' < \sigma_{\text{Total}}^{\text{soft}} / (\sigma_{\text{Total}}^{\text{soft}} + \sigma_{\text{Total}}^{\text{hard}}) \quad (3.3)$$

we are in the soft region, otherwise, we are in the hard region.



a) vertex corrections



b) Vacuum polarization



a) Initial state radiation



c) Pure QED box diagrams



b) Final state radiation



d) γ and Z_0 box diagrams

Fig. 4 Virtual corrections to the process $e^+e^- \rightarrow \mu^+\mu^- (\gamma)$

Fig. 5 Bremsstrahlung diagrams contributing to the cross section in the process $e^+e^- \rightarrow \mu^+\mu^- (\gamma)$

In the soft region, we can use as approximate cross section the lower order cross section in eq. 3.2, and neglect the momentum of the soft photon. This allows a very fast and easy generation. Only two variables have to be generated: the angles (θ, φ) between e^+ and μ^+ . φ is uniformly distributed in $(0, 2\pi)$ and $\cos \theta$ is distributed according to equation 3.2, in the interval $(-1, 1)$. The momentum of the outgoing μ 's is fixed,

$$p_\mu = \sqrt{E_{\text{beam}}^2 - M_\mu^2}.$$

In the hard bremsstrahlung region it is not longer possible neglect the photon. Then, the final state is composed by 3 particles, namely, γ , μ^+ and μ^- , characterized by their fourmomenta (12 variables), that should satisfy the energy-momentum conservation constraints (4 equations). There are then 5 independent variables to be generated according to the cross section in the hard bremsstrahlung region. This cross section can be denoted by the initial and final state radiation amplitudes

$$|M_i + M_f|^2 = |M_i|^2 + |M_f|^2 + 2 \operatorname{Re} (M_i M_f^*) \quad (3.4)$$

In order to generate events, it is possible to use, as approximate cross section, the sum of two positive terms: the initial state and the final state radiation cross sections. Then, the exact value of the cross section is in the region

$$0 \leq d\sigma^{\text{exact}} \leq 2d\sigma^{\text{appr}} \quad (3.5)$$

The procedure to generate events, using the importance sampling method, can be the following:

- 1) Generate variables $\Omega'_\mu, \Omega'_\gamma, K$ according to approximate cross section (procedure described in reference [9]).
- 2) Generate a number w' uniformly distributed in $(0, 2)$.
- 3) Accept $\Omega'_\mu, \Omega'_\gamma, K$ as good event if $w' \leq d\sigma^{\text{exact}} / d\sigma^{\text{appr}} (\Omega'_\mu, \Omega'_\gamma, K)$.
- 4) Repeat points 1, 2, 3 until an event is accepted.

3.1.2 QCD

QCD is well known from the computational point of view up to α_s^2 order. The possible parton configurations in e^+e^- annihilations, are

$$e^+e^- \rightarrow q\bar{q}, q\bar{q}g, q\bar{q}gg, q\bar{q}q\bar{q} \quad (3.6)$$

Although at the parton level, the problem of generating events is rather similar to the previous one, there are additional complications due, mainly, to the unknown fragmentation process. The fragmentation process, has to be modeled with some reasonable scheme predicting the production of particles in the final state. Here, as an example, the Feynman - Field approach is adopted to fragment partons.

To simplify this generator, only the QED initial state radiative corrections are included.

In the generation of this kind of events, the following scheme is taken

1. Generation of the initial state photon
2. Production of 2, 3 or 4 partons
3. Fragmentation

Generation of the initial state photon

At this point our problem is similar to that of the μ -pairs, with an additional complication: Our final state at the quark level is a mixture of $u\bar{u}$, $d\bar{d}$, $s\bar{s}$, $c\bar{c}$ and $b\bar{b}$ pairs. We must choose, at random, the kind of flavor to produce, according with its charge squared and then generate the radiative γ . The procedure to generate the radiative γ is described in reference [10]. A value K_{\min} , is choosen to separate soft and hard bremsstrahlung regions, in such a way that the photon energy can be neglected below K_{\min} , and another value K_{\max} to fix the maximum photon energy in the hard bremsstrahlung region.

In the soft region the cross section can be computed by integration of

$$\frac{d\sigma_{\text{had}}^{\text{rad}}}{d_{q_1 \dots q_n}} = \frac{d\sigma^0(s)}{d_{q_1 \dots q_n}} (1 + \delta_A + \delta_\mu + \delta_\tau + \delta_{\text{had}}) \quad (3.7)$$

where δ -corrections come from the soft bremsstrahlung, vertex and vacuum polarization correction, $q_1 \dots q_n$ is a convenient set of independent variables defining the final state and $\sigma^\circ(s)$ is the hadronic cross-section at energy \sqrt{s} .

In the hard region the total cross section can be computed from

$$\frac{d\sigma_{had}^{rad}}{dK d\Omega_\gamma} = \frac{\alpha K}{\pi^2 s} \left[\frac{m^2 s'}{4K_-^2} + \frac{m^2 s'}{4K_+^2} + \frac{s^2 + s'^2}{8K_+ K_-} - 1 \right] \sigma^\circ(s') \quad (3.8)$$

where K and Ω_γ are the photon energy and solid angle and

$$s' = Q^2 = 4 \frac{\sqrt{s}}{2} \left(\frac{\sqrt{s}}{2} - K \right)$$

$$K_\pm = P_\pm \cdot K$$

P_+ , P_- denote the four momenta of the e^+ , e^-

Integration of equation 3.8 in the Ω_γ variables leads to

$$\frac{d\sigma_{had}^{rad}}{dK} = \frac{\alpha}{\pi} \sigma^\circ(s') \left(\ln \frac{s}{m_e^2} - 1 \right) \left[1 + \left(\frac{s'}{s} \right)^2 \right] \frac{1}{K} \quad (3.9)$$

With the help of equations 3.7 and 3.9, after integration, we choose the region. In the soft region, we use $d\sigma^\circ(s) / dq_1 \dots dq_n$ to generate partons. In the hard region, we generate first the energy K and then, the solid angle of the photon by using equations 3.8 and 3.9 and finally the partons with $d\sigma^\circ(s') / dq_1 \dots dq_n$.

Production of 2, 3 or 4 partons

The evolution of the QCD matrix elements with the energy is known up to α_s^2 . Then we can generate 2, 3 or 4 partons with $d\sigma^\circ(s') / dq_1 \dots dq_n$ as was outlined above. This integrated cross section has the form:

$$\sigma^\circ(s') = \sigma^\circ_0 \left[1 + \frac{3}{4} C_F \frac{\alpha_s}{\pi} + K_{Ms} \left(\frac{\alpha_s}{\pi} \right)^2 \right] \quad (3.10)$$

Imposing "jet criteria resolution", we can construct the 3 and 4 jet production cross section to be finite

$$\sigma_{3\text{jet}} = \sigma_{3\text{ born}} + \sigma_{3\text{ virt}} + \sigma_{4\text{ soft}} \quad (3.11)$$

$$\sigma_{4\text{ jet}} = \sigma_{4\text{ born}} - \sigma_{4\text{ soft}} \quad (3.12)$$

The part of the total cross section which is neither $\sigma_{3\text{ jet}}$ or $\sigma_{4\text{ jet}}$, is called $\sigma_{2\text{ jet}}$

$$\sigma_{2\text{ jet}} = \sigma^\circ - \sigma_{3\text{ jet}} - \sigma_{4\text{ jet}} \quad (3.13)$$

A fraction of the final states with 3 and 4 partons can contribute to $\sigma_{2\text{ jet}}$ and $\sigma_{3\text{ jet}}$ when one of the partons has very low energy or is very close to some other parton, such that they fail the jet resolution criteria (see figure 6).

We choose the Sterman-Weinberg variables δ, ϵ to define the distinction level between close jets and the minimum energy required to define a jet.

$$\epsilon = \min \left\{ \frac{E_i}{s'} \right\}; \quad E_i = \text{Energy of parton } i \quad (3.14)$$

$$\delta = \min \left\{ \frac{\theta_{ij}}{2} \right\}; \quad \theta_{ij} = \text{Angle between parton } i \text{ and } j$$

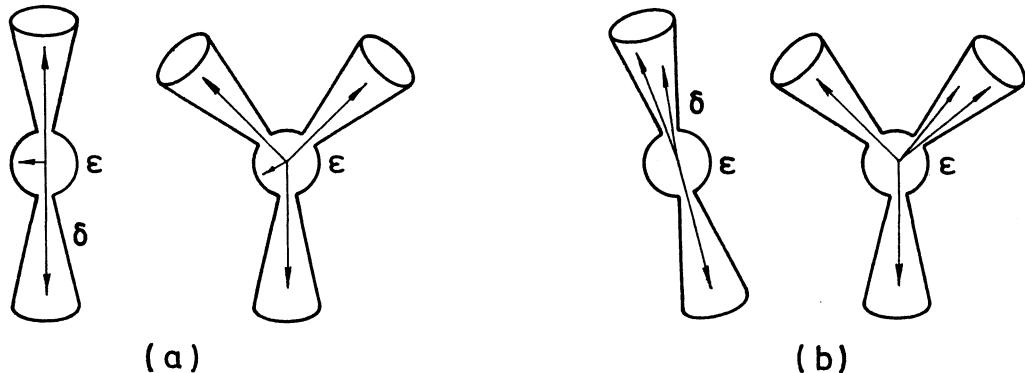


Fig. 6 Example of 3 and 4 partons failing the jet resolution criteria on the Sterman-Weinberg variables ϵ, δ and passing to the category of 2 and 3 jets. a) failing ϵ . b) failing δ .

Under these assumptions we can compute the total cross section for the production of 2, 3 and 4 jets independently, with radiative corrections included by numerical integration, using them as weight to choose how many jets to produce. [11].

Fragmentation

The process through which a parton produces other particles is called fragmentation. Fragmentation can not yet be calculated in QCD and then some parametrization for the production of final state particles, must be made [12]. Fragmentation should reproduce two experimental facts.

1. Reproduction of energy distributions of particles in a jet
2. Reproduction of the spread of particles in a jet.

The Feynman-Field fragmentation model describes the hadron production through the fundamental transition

$$q_a \rightarrow M(q_a, \bar{q}_b) + q_b \quad (3.15)$$

in which the incident quark q_a fragments into a meson M and a residual quark q_b . The process is repeated many times until the energy of the last residual quark is not enough to yield a new meson by picking up a pair $q\bar{q}$ from the sea, in which case the process is stopped. Two distributions characterize the formation of mesons in the process. The first is the fraction of the quark energy carried by the meson, described by a probability function $f(z)$, where

$$z = \frac{(E + P//)\text{meson}}{(E + P//)\text{quark}} \quad (3.16)$$

can be interpreted as the fraction of the energy of the quarks carried out by the meson. The second is the transversal spread of mesons described by an exponential probability function,

$$S(P_T) = \exp(-P_T^2 / 2\sigma_q^2) \quad (3.17)$$

where σ_q is a parameter to be fixed. The probability function $f(z)$ is normalized such that $\int_0^1 f(z) dz = 1$ and it is conventionally parametrized as

$$f(z) = 1 - \alpha_F + 3\alpha_F^2 (1 - z)^2 \quad (3.18)$$

although special shapes can be given, as is normally the case for heavy quarks.

The description of the basic process (3.15) is given by the function

$$D(z) = \frac{1}{\sigma_{\text{tot}}} \frac{d\sigma}{dz} \quad (3.19)$$

which is normalized such that

$$\int dz D(z) = \langle N(\pi) \rangle \quad (3.20)$$

where $\langle N(\pi) \rangle$ is the average number of π 's produced in the reaction.

$D(z)$ and $f(z)$ are related by the integral equation

$$D(z) = f(z) + \int_z^1 d\eta [f(1 - \eta) D(z/\eta)] \quad (3.21)$$

There are still two parameters coming from experimental data giving the rate of pseudo-scalar to vector mesons, a_v and the rate of strange particles production, a_s . The standard values given to the free parameters of the model are $a_v = 0.5$, $a_s = 0.4$, $\sigma_q = 300$ MeV and $\alpha_F = 0.77$.

The Feynman-Field model must be extended to fragment gluons. Here we adopt the extension proposal by Ali et al. in which the gluon is splitted in a $q\bar{q}$ pair according to the Altarelli-Parisi splitting function for $g \rightarrow q\bar{q}$

$$\frac{dp}{dz} \sim [z^2 + (1 - z)^2] \quad (3.22)$$

and then every quark is fragmented independently.

3.2 The detector simulation

A basic need for experimentalists is the ability of measuring properties of particles appearing in the final state of a physical process. This need is fulfilled with detectors, i.e. different materials with some convenient behaviour depending upon the particles crossing through. We can distinguish between two big sets of materials. Active and passive materials. Active, are those producing some kind of signal which is possible to pick up through some

interfase. Examples are the scintillator materials, gases with properties as the ability of being ionized or producing light when excited with particles passing through, or even gases in metastable state, like liquid Hydrogen in bubble chambers. There are many properties of materials that we can profit of. Passive materials are those which do not produce a direct signal to observe, but can help to some active material or can constitute a bulk for some particle difficult to cross or are the support of the detector components.

The detector simulation has two major parts: The simulation, as good as possible, of all the interactions, that a particle could have along the detector materials, like multiple scattering, nuclear interactions, coulomb interactions, etc, as well as, the subsidiary interactions of particles produced before. In here the development of the electromagnetic and hadronic shower constitute the goldstone of the simulation. The simulation of the signals coming from the active materials and further storage of them, usually in digital form, is the other major item.

3.2.1 Simulation of showers

By shower development we designate a set of phenomena involving transport and interactions of particles in media. A particle going through a medium has a path without interacting until it has some kind of interaction producing other particles which can, as well, have some free path to further interact and repeat the previous process until it ends up with a cloud of very low energy particles, most of them photons, that are absorbed by the media or are conveniently driven to some device to collect a signal.

The process of simulation of showers, will be performed with some basic ingredients: the simulation of the free path of particles, the simulation of the interactions and the accounting of energy that particles give to matter.

In relation with the relevant kind of interaction, we distinguish between "Electromagnetic and hadronic showers".

3.2.2 Electromagnetic showers

Two basic processes dominate the energy losses of an electron or positron when traversing matter: collision and radiation. The first has two possible effects; the atom is left in a excited state or is ionized. If in excited state, the atom emits a low energy photon to recover the ground state. If ionized, the ejected electron, has most of the time small energy which is locally deposited. However some time the orbital electron gets enough energy to be regarded as a secondary particle called Delta ray. In the case of radiation, the energy spectrum of the radiated photon goes from "0" to the maximum energy of the electron or positron. The collision process dominates at low energy, whereas radiation does at high energy. The radiated photon can it self interact with matter. Three photo-processes dominate, dependig on the energy of the photon and the nature of the medium: pair production or materialization, compton scattering and photo-electric effect. The first dominates at high energy whereas compton scattering does at lower energy. These two processes give back electrons to the system which can again radiate photons repeating and multiplying the process which is globally called "Electromagnetic cascade shower". The photo-electric effect as well as the Coulomb scattering of electrons introduce some perturbation in the shower. The shower lateral spread comes mainly from Coulomb scattering and Compton scattering. The net effect

in the development of the shower is an increase in the number of particles and a decrease in their average energy reaching energies low enough, so that collision losses become predominant.

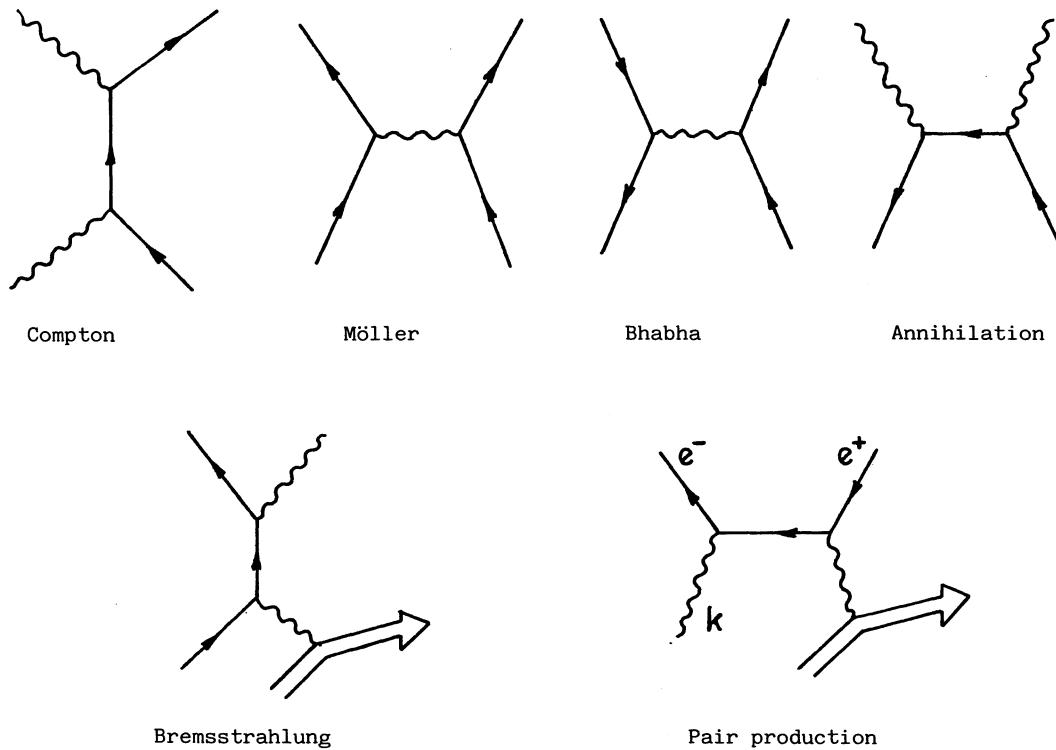


Fig. 7 Diagrams representing basic processes treated in the EGS code

In this paragraph, I will describe, briefly, one of the codes that, so far, is recognized to simulate with great accuracy the development of Electromagnetic showers. It is called EGS (Electron-Gamma Simulation).

The simulation of the electromagnetic shower is decomposed into a simulation of the transport and interactions of a single particle. The information about the particles produced at the interaction is stored in the top of a stack initiated with the primary particle of the shower. The basic strategy is to transport the particle in the top of the stack until an interaction takes place, or until its energy drops below some given cutoff energy, or until it enters a particular space region. In the later two cases the particle is taken out of the stack and simulation resumes with the next top particle.

Transport

Let σ_T be the total cross section per molecule of a particle in a medium, then, the mean free path, $\bar{\lambda}$, is given by

$$\bar{\lambda} = \frac{A}{N_a \rho \sigma_T} \quad (3.23)$$

where A , N_a and ρ are the molecular weight, Avogadro's number and density respectively.

In general the mean free path may change as the particle moves from one medium to another, or when it losses energy. The number of mean free paths crossed when going from a point x_0 to x will be

$$N' \lambda = \int_{x_0}^x \frac{dx}{\lambda(x)} \quad (3.24)$$

It can be shown that the probability of a particle to traverse $N' \lambda$ mean free paths before interaction is

$$G(N' \lambda) = e^{-N' \lambda} \quad (3.25)$$

Obviously this expression gives the possibility of sampling random values of $N' \lambda$ since the function G is distributed in $[0, 1]$

$$N' \lambda = -\ln G' \quad (3.26)$$

This value can be replaced in equation (3.24) to obtain the location of next interaction point.

Transport of photons

The only effects accounted for photons in EGS are: Pair production, Compton scattering and photoelectric processes. The cross sections for the above processes are finite and small enough that all interactions can be simulated. The photon travels along a straight line with constant energy between interactions. Assuming that the medium in which the simulation takes place is composed of a finite number of homogeneous materials with constant density, then the integral 3.24 becomes

$$N' \lambda = \sum_{j=1}^{i-1} \left(\frac{x_j - x_{j-1}}{\lambda_j} \right) + \frac{x_i - x_{i-1}}{\lambda_i} \quad (3.27)$$

where $x \in (x_{i-1}, x_i)$ and x_0, x_1, \dots are the boundary distances between materials with λ constant.

Simulation of photon transport is then as follows: First, pick up the number of mean free paths to the next interaction, $N' \lambda$, using 3.26. Then perform the following steps

- 1) Compute λ at the current location
- 2) Let $t_1 = \lambda N' \lambda$
- 3) Compute d , distance to next boundary along the photon direction
- 4) Let $t_2 = \text{the smaller of } t_1 \text{ and } d$. Transport by distance t_2
- 5) Subtract t_2/λ from $N' \lambda$. If the result is zero (this happens when $t_2 = t_1$), then, the interaction takes place. Jump out of the loop to treat the interaction

6) Here we arrive when $t_2 = d$. Thus, a boundary was reached. If the new region is a different material go to step 1, otherwise to step 2.

If any of the particles resulting in the interaction is again a photon, and is still inside the medium where tracking is possible a new number N_{λ} is chosen and steps from 1. to 6. are performed.

Transport of charged particles

Several problems appear in the transport simulation of electrons and positrons. The relevant interactions considered are elastic Coulomb scattering off the nucleus, inelastic scattering off the atomic electrons, bremsstrahlung production and positron annihilation into photon pairs. The first three processes have a very large cross section resulting in the production of a cloud of low energy particles which is not practical to simulate in a discrete way. For this reason we shall apply cut-off energies to distinguish between continuous and discrete interactions. In this way when in a given interaction the energy of the electron produced is above a given cut-off AE or the energy of the photon is above AP, we consider them as discrete events, otherwise they are treated in a continuous manner. All this particles in the continuum give rise to energy losses and direction changes of the electron between discrete interactions. The energy losses are due to soft interactions with the atomic electrons (ionization loss) and to the emission of soft bremsstrahlung photons. The changes in direction are mostly due to multiple Coulomb scattering from the nucleus, with some contribution coming from soft electron scattering.

The energy losses and the changes of direction along the path of electrons and positrons complicate the transport simulation, since the cross section changes along the path and the paths are no longer straight lines. The fact that the discrete electron total cross section decreases with decreasing energy allows the following trick which is used to account for the change of λ along the path: We introduce a fictitious interaction such that the total cross section is constant along the path

$$\sigma_{T,fict}(x) = \sigma_{T,real}(x) + \sigma_{fict}(x) = \text{constant} = \sigma_{T,real}(x_0) \quad (3.28)$$

The location of the "next interaction" is then sampled using, as before, equations 3.26 and 3.24, along with the total fictitious cross section $\sigma_{T,fict}(x)$. When the interaction point is reached, we generate a random number between 0 and 1 and if it is greater than $\sigma_{T,real}(x)/\sigma_{T,real}(x_0)$, then, the interaction is fictitious and the transport continues from that point without interaction, otherwise the interaction is real.

The remaining problem is the change of direction between interaction points due to multiple scattering. We shall give here a general explanation. The transport of the electron between interactions is divided into smaller steps which are supposed to be straight lines. The multiple scattering is accounted for by changing the electron direction at the end of each step. The angle between the initial and final directions is sampled from the appropriate distribution and the azimuthal angle is chosen randomly.

Interactions

Here we shall describe the general procedure to decide what interaction we deal

with at a given interaction point.

The probability that a given type of interaction occurs is proportional to its cross section. Suppose that all possible types of interactions at a given point are numbered from 1 to n. Let's divide the interval $[0, 1]$ in n ordered subintervals i , (a_{i-1}, a_i) such that

$$a_i - a_{i-1} = \frac{\sigma_i}{\sigma_T} \quad (3.29)$$

Then, we decide that the interaction is of type i , if by choosing a random number $r \in [0, 1]$, it satisfies

$$a_{i-1} < r \leq a_i \quad (3.30)$$

Once the type of interaction has been decided, the next step is to determine the parameters of the produced particles. (In general the four-momentum of particles). Suppose that the final state is characterized by, say, n parameters q_1, q_2, \dots, q_n . The differential cross section will have an expression of the form

$$d^n \sigma = g(\vec{q}) d^n q \quad (3.31)$$

with a total cross section given by

$$\sigma = \int g(\vec{q}) d^n q \quad (3.32)$$

The function

$$f(\vec{q}) = g(\vec{q}) / \sigma \quad (3.33)$$

is normalized to 1 and has the properties of a probability density function. This may be sampled by using some of the methods given in section 2.3 and then obtain the parameters q_1, q_2, \dots, q_n . The previous parameters define the particles produced as result of the interaction. We store them in the stack and we follow with the simulation taking the particle in the top of the stack.

The discrete interactions which are treated in detail in the EGS code are

- Photons interacting (Compton scattering, pair production and photoelectric effect)
- Electrons or positrons interacting (Multiple scattering, annihilation, bhabha scattering, Möller scattering and bremsstrahlung).

To finish this paragraph we shall comment on the continuous electron energy loss. As we said, the continuous loss is the result of interactions in which the energy transfer to secondary particles is not enough to put them above the discrete transport energy thresholds. The secondary particles are either soft breemstrahlung photons, or atomic electrons which absorb some energy. The mean total continuous energy loss per unit length is

$$- \left(\frac{dE}{dx} \right)_{\text{Total continuous}} = - \left(\frac{dE}{dx} \right)_{\text{Soft Bremsstrahlung}} + - \left(\frac{dE}{dx} \right)_{\text{Sub-cutoff Atomic electron}} \quad (3.34)$$

where $-$, $+$ denotes electron or positron.

In this code it is supposed that when an electron is transported by a given distance, its loss of energy due to sub-cutoff secondaries is equal to the distance travelled times the mean loss per unit length as evaluated using previous equation. Actually, energy loss over a transported distance is subjected to fluctuations and gives rise to a Landau distribution (Not implemented yet).

3.2.3 Hadronic showers

The simulation of hadronic showers, although has some similarity with the one considered in the previous section, has different features, due mainly to the variety of particles and interactions involved in it. In here, we understand the hadronic shower as the development of interactions in cascade with matter, carried mainly by hadrons, and with the starting particle being a hadron. The transport of particles is carried step by step computing the interaction probability in each, based on the total cross section.

There are many processes involved in the development of hadronic showers: on one side, we have the energy loss, γ -ray emission and multiple scattering of the particle along its way, on the other side, we have the nuclear scattering of this particle in media with problems concerning to the final state of the collisions, such as, the generation of multiplicity distributions, energy distributions, angular distributions, etc. In addiction, if as a result of the nuclear scattering, there is production of neutral pions, these will decay to photons, starting electromagnetic cascades.

In this section, we describe one of the codes (GHEISHA, [14]), which is being successfully used in the Mark-J experiment at PETRA, to simulate the passage of particles through the detector, and that is being used in the specific design of some of the components of the L3 detector at LEP.

Energy Losses

The ionization energy losses of charged particles in matter are carried using the notation of Sternheimer: The most probable energy loss is

$$-\frac{dE}{dx} = \frac{D}{\beta^2} \left[G + 2 \ln \frac{\beta^2}{1-\beta^2} - 2\beta^2 - 2\delta(\beta) \right] \quad (3.35)$$

where β is the velocity of a particle traversing an absorber with atomic number, atomic weight and mean excitation potential A, Z and I, respectively. D and G are functions of A, Z and I. $\delta(\beta)$ is the density effect.

The energy loss of a particle for a given length is subject to fluctuations. Under some conditions the energy loss distribution is close to a Gaussian distribution, under other conditions is close to a Landau distribution. In GHEISHA we use a Gaussian distribution when $W/E_{max} \gg 1$ and Landau when $W/E_{max} \ll 1$ where

$$W = 0.0001536 \rho \frac{Z}{A} x / \beta^2 \quad \text{and} \quad (3.36)$$

$$E_{max} = 2m_e \beta^2 / (1 - \beta^2) \quad (3.37)$$

In the region where $W / E_{\max} \sim 1$ we use for liquids and plastics Landau distributions and for metals Gaussian fluctuations. (Let us note than in the EGS code there were not such fluctuations).

δ -rays

The δ -rays are the electrons coming from the ionization of atoms in matter. The distance they will cover is approximated by

$$R_E = 0.71 E^{1.72} / \rho \text{ (cm)} \quad (3.38)$$

where E is given in MeV and ρ in g/cm³. The simulation of δ -rays is made in the following way. First we give the minimum range R_{\min} fixed normally by the position resolution of the detector. Then we compute E_{\min} by inverting the formula above. The number of δ -rays is Poisson distributed around $\langle N \rangle_E$ given by

$$\langle N \rangle_E = W / E_{\min} \quad (3.39)$$

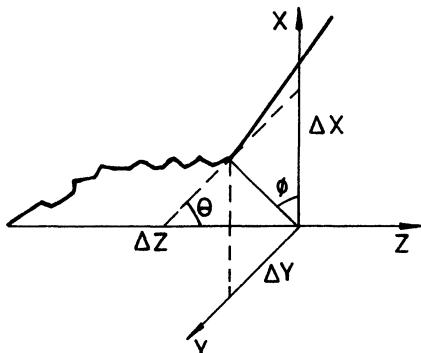
and the energies of the electrons are generated according to

$$P(E) dE = W \frac{dE}{E^2} \quad (3.40)$$

The emission angle θ with respect to the primary particle direction is given by

$$\cos^2 \theta = E / E_{\max} \quad (3.41)$$

The energy of δ -rays is subtracted from the energy deposit of the ionizing particle. It should be noted that the contribution from δ -rays is only meaningful in light materials.



Multiple scattering

For identification of muons, we normally use thick absorbers to stop hadrons which could be missidentified as muons. The use of absorbers, nevertheless, produces an increase in the angular divergence and lateral position of a particle traversing it. The simulation of the multiple small collisions that a particle has through a medium is described here. The average of the total deflexion squared angle in a step of length z is given by [15]

$$\langle \theta_x^2 \rangle = \left(\frac{\alpha}{p\beta} \right)^2 \frac{z}{x_0} \left[1 + \frac{1}{9} \log_{10} \left(\frac{z}{x_0} \right) \right]^2 \quad (3.42)$$

where $\alpha = 0.015 \text{ GeV}$, x_0 is the radiation length of the material and p is the momentum.

Then the simulation of multiple scattering in a small step is carried by, first generating

two numbers A and B distributed according to a two-dimensional Gaussian with r.m.s of 1 each. The angles $\Delta\theta$, $\Delta\psi$ in fig. 8 are

$$\sin(\Delta\theta) = \sqrt{\langle\theta_x^2\rangle} B \quad (3.43)$$

$$\sin(\Delta\psi) = \sqrt{\langle\theta_x^2\rangle} A \quad (3.44)$$

and the displacements Δx and Δy

$$\Delta x = \sin(\Delta\theta) \cdot \Delta z / 2 \quad (3.45)$$

$$\Delta y = \sin(\Delta\psi) \cdot \Delta z / 2 \quad (3.46)$$

The step size in the tracking procedure is rather small ~ 0.1 to 0.2 cm. The decrease of momentum due to the ionization energy loss is applied after the step has been carried out.

Nuclear scattering. Transport

The transport of particles produced in the hadronic shower is carried in a similar way to what was done in the previous chapter. Depending on the material, momentum and type of incoming particle we have an interaction probability given by

$$P(x) = 1 - \exp(x / \lambda) \quad (3.47)$$

where x is the length of the step and λ is the nuclear collision length given by

$$\lambda = \frac{A}{N_a \rho \sigma} \quad (3.48)$$

To see if there is interaction or not we sample in the distribution

$$x' = \lambda \ln \{ 1 - p' \} \quad (3.49)$$

where p' is random uniformly distributed in $(0, 1)$. If $x' \leq x$ then an interaction has occurred and we should choose the kind of interaction to perform, otherwise there is no interaction and we transport the particle a length x from the previous point.

Interactions

As quoted before, the bulk of interactions in hadronic showers are with nuclei. There are four major classes of interactions of hadrons with nuclei (see figure 9), depending on the status of the nucleus after the interaction:

- a) coherent elastic
- b) coherent inelastic
- c) Incoherent quasielastic
- d) Incoherent inelastic

The first two are well understood from the theoretical point of view and a generalization of the hadron-nucleon interactions can be applied to them. Formally the nucleon with $A = 1$ has to be replaced with $A > 1$. The incoherent scattering is more complicated and involves the modeling of the intranuclear cascade. As a further step we must take care of Nuclear

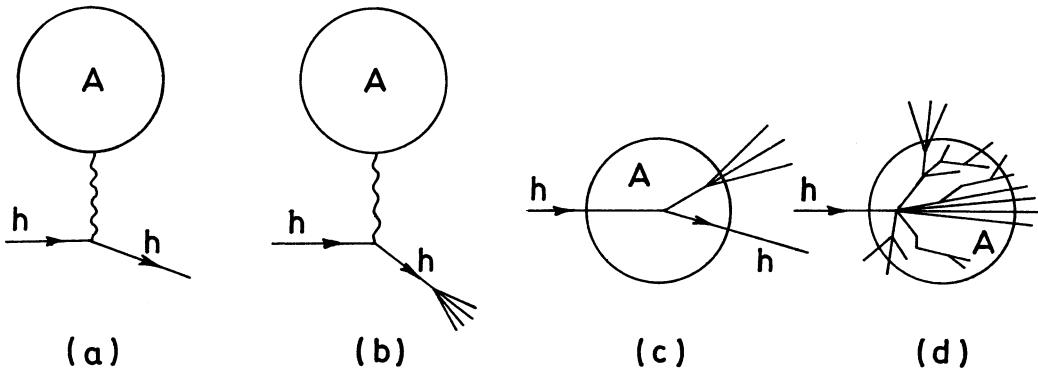


Fig. 9 Interactions hadron-nucleus a) coherent elastic, b) coherent inelastic, c) incoherent quasielastic, d) incoherent inelastic

evaporation, stopping particles and nuclear fission.

The elastic and inelastic cross sections hadron-proton are tabulated in GHEISHA, as a function of the momentum for π^+ , π^- , K^- , K^+ , K_L^0 , p and \bar{p} . For all the other cross sections on protons we use the charge independent approximation for the forward elastic scattering amplitudes according to the naive quark parton model and write them as function of the quark scattering amplitudes. By using the available data we fit the quark scattering amplitudes as a function of the momentum and determine the corresponding cross sections of all unmeasured or poorly measured reactions. The cross sections for target neutrons are assumed to be the same as for protons.

It is commonly accepted that diffraction and absorption cross sections for the scattering on heavy nuclei can be parametrized as

$$\sigma_d = \sigma_{do} A^{\alpha_d} \quad (3.50)$$

$$\sigma_a = \sigma_{ao} A^{\alpha_a} \quad (3.51)$$

where A is the atomic number. The formulas are based in the optical model, that assumes that at high energy the nucleus structure can be approximately described by an average potential. The real and imaginary part of the potential gives the diffraction and absorption cross sections. We identify the absorption cross section with the coherent elastic scattering whereas the coherent inelastic and incoherent scattering contribute to the absorption cross section.

The parameters σ_{do} , σ_{ao} , α_d , α_a are determined by fits to the A -dependence of cross sections coming from the available experimental data. Nevertheless the program uses the

experimental data when it is available. In the very low energy region, $p \leq 0.5 \text{ GeV}/c$, the above parametrization has problems. This region is where the nucleus gets into an excited state and emits heavy particles like p, d, t etc (Nuclear reactions become dominant). The formulas 3.50 and 3.51 can no longer describe the cross sections in this region. To avoid these problems is quite wise to measure all the elastic and inelastic cross sections of all the materials used in an experimental set-up and insert these values in the program.

For the cross sections of the scattering on molecules and compounds, shadowing effects may play an important role and a direct scaling based on the atomic number of the constituents may not be right. The right definition of the effective atomic number lies somewhere between the average atomic number and the sum of the atomic numbers of all the constituents.

Generation of secondary particles

The generation of secondary particles in inelastic reactions hadron-nucleon is carried out by the special Koba-Nielsen-Olesen (KNO) formula

$$P(n) = \frac{1}{\langle n \rangle} \frac{n}{\langle n \rangle} \exp \left| \frac{\pi}{4} \left(\frac{n}{\langle n \rangle} \right)^2 \right| \quad (3.52)$$

where $\langle n \rangle$ is the average number of particles produced at a given energy. A fit to the experimental data in pp interactions gives for $\langle n \rangle$,

$$\langle n \rangle = 3.626 + 0.666 x + 0.337 x^2 + 0.118 x^3 \quad (3.53)$$

where $x = \ln (\sqrt{s} - 2m_p)$.

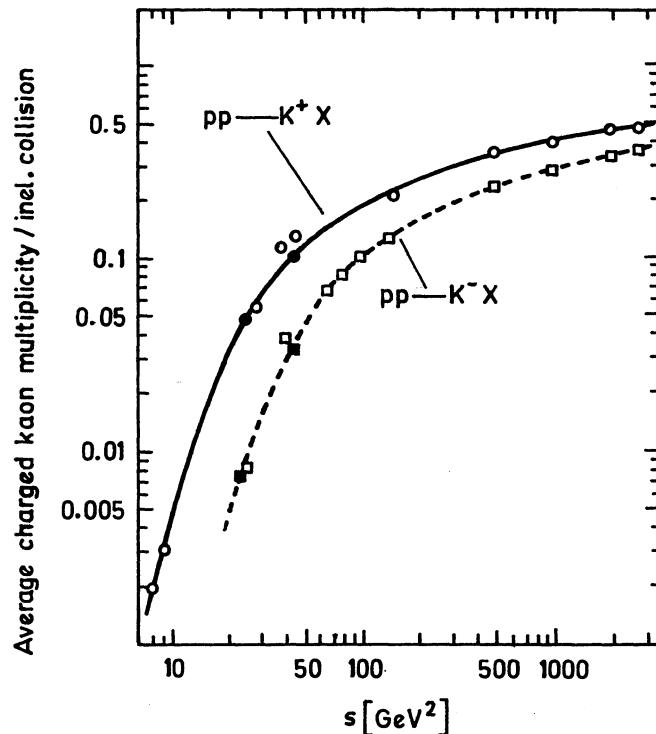


Fig. 10 Average charged kaon multiplicity per inelastic collision as a function of s . [16]

The multiplicity distribution for the production of n_{π^+} positive pions, n_{π^-} negative pions, n_{π^0} neutral pion, n_{K^+} positive kaons, etc, can be expressed by

$$W(n_{\pi^+}, n_{\pi^-}, n_{\pi^0}, n_{K^+}, n_{K^0}, \dots) = \\ = P(n) \cdot P(n_{\pi^+}, n_{\pi^-}, n_{\pi^0}, n_{K^+}, n_{K^0}, \dots | n) \quad (3.54)$$

where the conditional probability $P(n_{\pi^+}, n_{\pi^-}, \dots | n)$ can be obtained from the isospin model and tables. Three constraints are imposed in the produced particles : 1.) charge conservation, 2.) Strangeness conservation and 3.) Baryon number conservation. In figure 10 we show the average charged kaon multiplicity per inelastic collision for pp collisions as a function of s .

Simulation of final state kinematics

The generation of the four-momenta for the final state particles is based on quark or hadron-jets in soft hadron-nucleon collisions. As discussed in reference [14] it is possible to find scaling laws for the flux of energy and additive quantum numbers into a given interval of the polar angle Θ with respect to the jet axis

$$\frac{d\epsilon}{d\lambda} = \rho(\lambda) = \frac{M}{(1 + M^2 \lambda^2)^{3/2}} \quad (3.55)$$

$$\frac{dQ}{d\lambda} = Q\rho(\lambda) \quad (3.56)$$

$$\frac{ds}{d\lambda} = S\rho(\lambda) \quad (3.57)$$

where the variables are defined as follows:

$$\lambda = \cot \Theta / E_{jet} \quad (3.58)$$

$$\Delta\epsilon = E(\Theta) / E_{jet} \quad (3.59)$$

$$\Delta Q = \langle Q(\Theta) \rangle \quad (3.60)$$

$$\Delta S = \langle S(\Theta) \rangle \quad (3.61)$$

$\Delta\epsilon$, ΔQ and ΔS are the energy fraction, average charge and average strangeness emitted into an angular interval Θ and $\Theta + \Delta\Theta$. Q and S are the charge and strangeness of the fragmenting quark. And M is the mass parameter which is interpreted as the average transverse mass. The M parameter has different values for each fragmenting quark.

Choosing a momentum distribution of the quarks inside the hadron one can calculate the corresponding energy and quantum number flow distributions for pion and proton fragmentation.

To finish with the inelastic hadron-nucleon cross sections we summarize the procedure to obtain the final state particles. We stay in the c.m.s. of the colliding hadrons.

1. Calculate number of secondary particles and assign mass, baryon-number, charge and strangeness according parametrization 3.54.
2. Generate P_T -values and φ -values (azimuth angle) for all the particles, so that the transverse momenta balance exactly in the plane perpendicular to the beam axis. The

P_T distributions are generated according to $\exp(-b P_T^2)$, where b is tabulated from experimental data.

3. Assign λ -values in $[0, 1 / P_T]$ according to the three fragmentation equations 3.55 to 3.57, where Q and S are the charge and strangeness of the fragmenting beam or target particle. The mass M is chosen slightly different for pions, kaons and baryons.
4. Calculate x according $x = \lambda \cdot P_T$ and $P_L = P_0 \cdot x$
5. Choose a scale factor for the longitudinal momentum conservation. This factor is normally very close to one.

Elastic scattering and quasi-two-body reactions

Elastic scattering and quasi-two-body reactions are generated both in the same way. The production of secondaries with masses M_1 and M_2 obeys the law

$$\frac{d\sigma}{dt} \sim e^{-b|t|} \quad (3.62)$$

where t is the momentum transfer in the limits between t_{\min} and t_{\max} . The slope parameter can be parametrized as

$$b = \begin{cases} 4.225 + 1.795 \ln p & (\text{elastic}) \\ 4.000 + 1.600 \ln p & (\text{quasi-two-body}) \end{cases} \quad (3.63)$$

where p is the beam momentum in the Lab. system.

The masses M_1 and M_2 are distributed like

$$P(M) = g c [g(M-M_0)]^{c-1} \exp [-[g(M-M_0)]^c] \quad (3.64)$$

where M_0 is the threshold mass

$$M_0 = \sum_{i=1}^N m_i \quad (3.65)$$

and g , c are two parameters which determine the shape of the distribution.

Intra-nuclear cascade. Nuclear evaporation

In high energy hadron-nucleus collisions the nucleus can be considered as a dense set of nucleons. Nuclear structure, correlations between nucleons can be ignored. That comes from the fact that the wavelength of the incoming high energy hadrons is much smaller than the nuclear radius and typical times of hadronic interactions are shorter than nuclear periods.

The absorption length for protons in nuclear matter is ~ 2 fm. which is smaller compared, for instance, to the Uranium diameter ~ 14 fm.

Two situations can be considered depending on the distance in which an interaction hadron-nucleon inside the nucleus yields the asymptotic final state. If this distance is less than the typical dimension of the nucleon, then the final state particles are produced

inside the nucleus with possible further interactions giving rise to a "intranuclear cascade". The final number of particles leaving the nucleus is given by

$$\langle n \rangle_A = \langle n \rangle_p A^{1/3} \quad (3.66)$$

When the typical distance for the production of the asymptotic final state is bigger than the nuclear radius, the asymptotic hadrons will be produced after the nucleus, through, perhaps, an excited hadron state. In this case the multiplicity is

$$\langle n \rangle_A = \langle n \rangle_p \quad (3.67)$$

In other words we can say that backward produced particles in the center of mass of the first collision, do make an intranuclear cascade, whereas forward produced particles do not.

The first collision is exactly simulated in the program. The forward particles produced are not furtherly affected. The backward produced particles undergo a further intranuclear cascade. The average number of particles additionaly produced is parametrized with the formula

$$\langle n_{\text{add}} \rangle = \alpha(s, A, n) (A^{1/3} - 1) 2 n_b \quad (3.68)$$

where n_b is the number of backward produced particles and $\alpha(s)$ is a very slow varying function of s . n_{add} is poisson distributed.

In the phenomenology of nuclear interactions, the particles produced in the previous interactions are called shower particles (mesons and leading baryons) and grey track particles (nucleons from cascading processes).

The nuclear evaporation shows that after the interaction, the nucleus remains in a highly excited state emitting further nucleons, low mass fragments like d, t, α -particles and also high mass fragments. These particles are called black track particles. The kinetic energy is very low (< 20 MeV) and the angular distribution is isotropic in the laboratory system. The number of black particles is nearly proportional to the number of grey track particles

$$\langle n_b \rangle = 1.50 + 1.35 n_g \quad (3.69)$$

The total amount of kinetic energy communicated, to the particles in the nuclear evaporation is in first approximation

$$E_{\text{eva}} = F(A) \cdot G(E_k) \quad (3.70)$$

where $F(A)$ and $G(E_k)$ are functions of the atomic number A and of the kinetic energy, E_k of the primary particle.

Nuclear fission

Four steps can be distinguished in the fission process (suppose we have Uranium)

1^o) nuclear interaction with production of 2 heavy fragments ($\tau \sim 10^{-24}$ sec.)

2^o) deexcitation of the fragments by neutron emission ($\tau \sim 10^{-14}$ sec.)

3º) deexcitation by photon emission ($\tau \sim 10^{-8}$ sec.)

4º) deexcitation by β -decay $\tau \sim 10^{-6}$ sec.).

The energy spectra of the emitted neutrons has been parametrized as

$$P(E_n) = e^{-E/0.965} \sinh \sqrt{2.29 E}, E \text{ in MeV.} \quad (3.71)$$

and the number of emitted neutrons is normal distributed around

$$N(E_o) = 2.569 + 0.559 \ln E_o \quad (3.72)$$

with a r.m.s. of 1.23. E_o is the energy of the incoming neutron in MeV.

The energies of the emitted photons are given by

$$P(E_\gamma) = 10 e^{-1.15 E_\gamma} \quad (3.73)$$

and the number of emitted photons

$$N(E_o) = 9.5 + 0.6 \ln E_o \quad (3.74)$$

with a r.m.s. of 3.

The Gheisha-Monte Carlo described so far, is sufficient for the description of showers in calorimetric experiments. All the formulas in it contained are based on experimental data when available. In some case the parameters entering in formulas describing some hidden processes had to be tuned to reproduce related distributions according with the data. Some topics, like decay of particles, have not been discussed here, although they are included in the code.

3.3 Response simulation

To close these lectures, the simulation of the response of a calorimetric detector will be presented, as well as some comparison with experimental data. In general a calorimeter has active and passive components. The active components can be materials with luminescent properties, or drift chambers or some other device producing some convenient signal. A scintillator is chosen as active material to study the simulation.

Several effects are accounted since particles go through the scintillator until a signal is conveniently stored in some device: 1) The conversion of absorbed energy into fluorescent light, 2) The conduction of light through scintillator and Light-guides, 3) The conversion of light into photo-electrons in the photomultiplier (P.M.), 4) Amplification inside the photomultiplier of the electronic cascade and further transformation into a electric current and 5) The current integration, pulse shaping and read-out by the analog-to-digital converter (ADC) and time-to-digital converter (TDC). All this effects must be accounted when simulating interactions since we want to compare with experimental data obtained in this way.

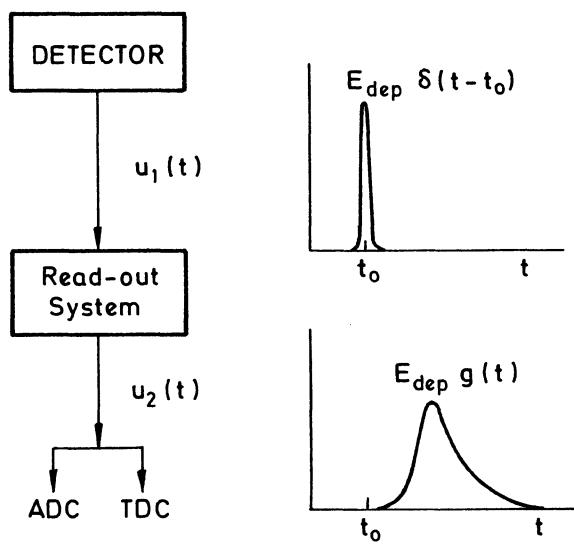


Fig. 11 The way that a signal follows since its production until an ADC or TDC is obtained. The read-out system is characterized by a response function $g(t)$

through (Birk's Law)

$$\frac{dL_o}{dx} = \text{const. } \frac{dE}{dx} / (1 + \alpha \frac{dE}{dx}) \quad (3.76)$$

where the light produced per unit of length is almost proportional to the kinetic energy loss. The parameter α obtained from fits to experimental data.

The atoms and molecules of the scintillator excited at a time t_0 by a primary event have a certain lifetime. Then, the intensity of the emitted light will decrease exponentially with the time

$$L_1(t) = \frac{L_o}{\tau_{fl}} e^{-\frac{t-t_0}{\tau_{fl}}} \cdot H(t - t_0) \quad (3.77)$$

where $H(t)$ is the step function with value "0" for $t < 0$ and 1 in all the other cases. Inorganic scintillators have normally a slower fluorescence decay time ($BGO \sim 320$ ns, $NaI(Tl) \sim 250$ ns., $ZnS(Ag) \sim 200$ ns.) than organic scintillators ($Anthracene \sim 32$ ns., $Stilbene \sim 4$ ns., $Liquid \sim 3$ ns., $plastic \sim 4$ ns.).

The light received at the entrance of the photomultiplier is given by

$$L_2(t) = \epsilon L_1(t - \frac{x}{c_{scin}} - \frac{d}{c_{LG}}) e^{-x/\lambda_{att}} \quad (3.78)$$

where ϵ is a constant accounting for light losses in the light guide, λ_{att} is the effective attenuation length combining effects like secondary excitation, light losses by diffuse reflections on surfaces, etc, x is the length of the light rays in the scintillator, d the

The read-out system to collect a signal can be described in general by a expression (see figure 11)

$$u_2(t) = \int_0^t u_1(t - \tau) g(\tau) d\tau \quad (3.75)$$

where $u_1(t)$ is the imput signal, $u_2(t)$ is the output signal and $g(\tau)$ is the response function which can be calculated or measured experimentaly. The response function involves all the effects that modify the imput signal until getting the output signal: fluorescence, decay of it, transport of light, attenuation, shape of devices, etc.

The amount of fluorescent light produced per unit of energy absorbed in the scintillator depends on the particle type, energy and the kind of material used as scintillator. The scintillation response can be calculated

length of the light guide and $C_{\text{scin}} (C_{\text{LG}})$ is the velocity of light in the scintillator (Light guide).

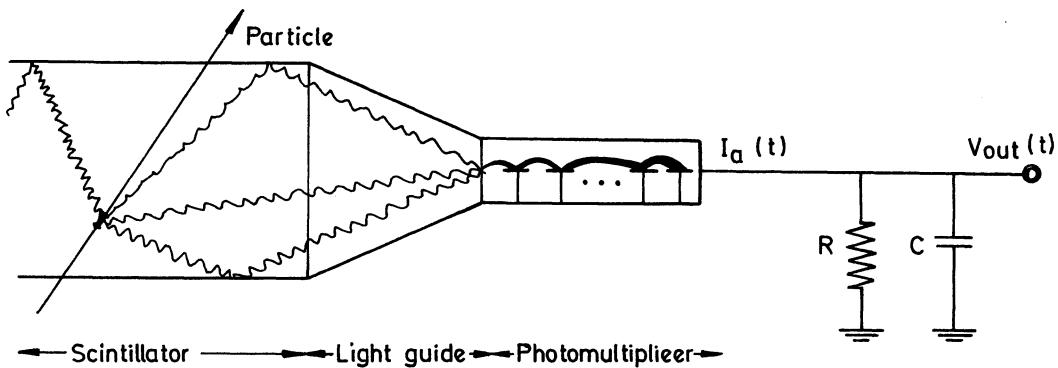


Fig. 12 Schematic view of the processes, since the scintillator is excited until a signal is collected.

After the interaction in the scintillator, the light is emitted isotropically, so that a distribution $F(x)$ of the light ray lengths x , must be folded into the above expression to get an effective signal at the entrance of the P.M.

Under the assumption, that the integration time constant RC of the output system is much greater than the pulse duration, and that the propagation time of the electron cascade in the PM and the spread of it are much smaller than the fluorescence decay time, we can get the anode current at output of the P.M.

$$I_a(t) = \bar{A} e \bar{N} L_2(t) \quad (3.79)$$

where \bar{A} is the mean amplifier gain, e the electron charge and \bar{N} the number of electrons reaching the multiplier system of the PM. The energy $W = \Delta E/\bar{N}$ necessary to form a photo-electron is relatively high. (0.3 – 1 KeV for NaI(TL), 1-3 KeV in organic scintillators, 5-20 KeV in glass scintillators)

The anode current $I_a(t)$ can be integrated over by a RC network to obtain a voltage pulse proportional to ΔE . The time dependence of the output voltage is

$$V_{\text{out}}(t) = - \frac{\bar{A} e \bar{N}}{C} \int_0^t e^{-(t-t')/t_{\text{RC}}} L_2(t') dt' \quad (3.80)$$

With time constants t_{RC} long enough, the voltage pulse height is given by

$$V_o = \frac{\bar{A} e \bar{N}}{C} \quad (3.81)$$

The deviation σ_v of the total pulse light V_o depends on the statistical deviation $\sigma_{\bar{N}}$ of the number \bar{N} of photo cathode electrons and on the deviation $\sigma_{\bar{A}}$ of the P.M. gain:

$$\left(\frac{\sigma_v}{V_o}\right)^2 = \frac{1}{\bar{N}} \left[1 + \left(\frac{\sigma_{\bar{A}}}{\bar{A}}\right)^2 \right] \quad (3.82)$$

The response function, $g(t)$ may be obtained by solving the integral (3.80) or (3.79) replacing $L_2(t)$ by (3.78) and $L_1(t)$ by (3.77).

Once the function $g(t)$ is known, the simulation of the output voltage can be easily performed by sampling the absorbed energy of the whole event in the scintillators connected to the same P.M., as function of the space and time F , t . F may be converted to x and the integral (3.75) gives immediately the output voltage as function of t .

Unfortunately, the response function $g(t)$ is not possible to solve if the distribution of light ray lengthes $F(x)$ is not simple enough.

A general way to skip this problem, is to produce n light rays at the position of the energy absorbtion applying a time delay for each ray according (3.77), then use (3.78) for the light absorbtion assuming some distribution function $F(x)$. For energy light ray i , $V_i(t)$ is calculated by numerical integration using (3.80). The total pulse shape is obtained by sampling all $V_i(t)$ in a histogram.

The Monte Carlo method has played and is playing an important role in the specific design of the components of detectors. Monte Carlo studies were made, for instance, to decide the optimal BGO crystal size in the L3 electromagnetic calorimeter at LEP.

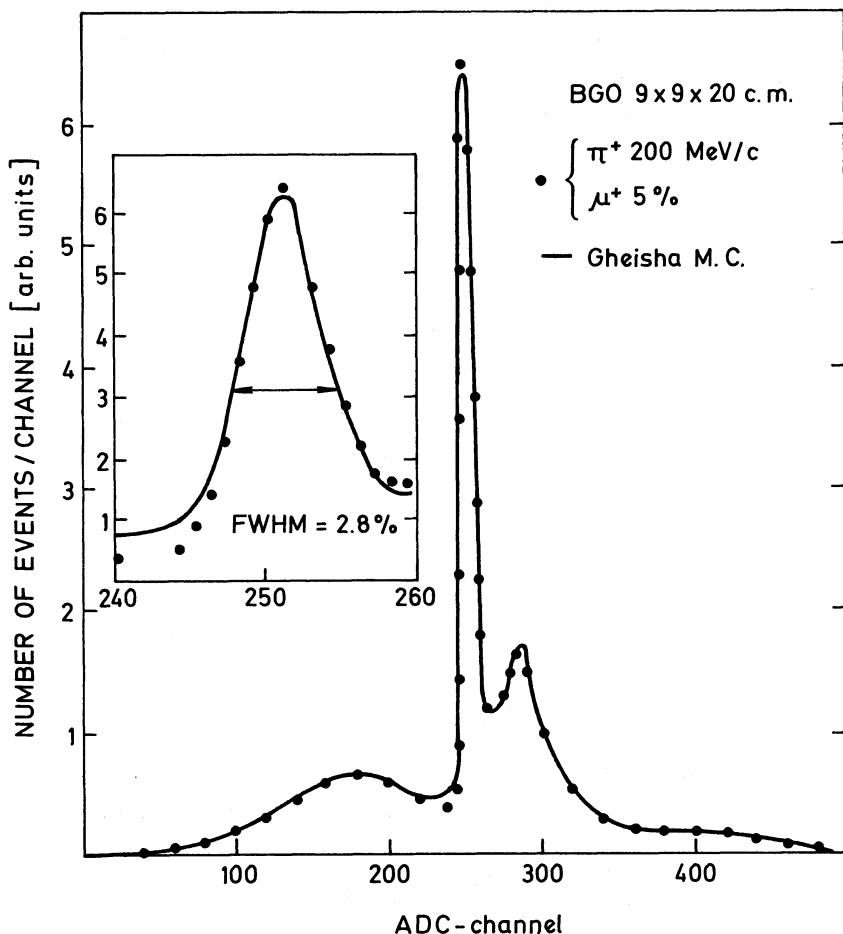


Fig. 13 Pulse hight distribution. Comparison between data and Gheisha Monte Carlo.

Test runs with e, μ , and π beams have been carried at CERN to test scintillator performances and readouts for the BGO electromagnetic calorimeter. Pion runs have been made with a BGO crystal size of $9 \times 9 \times 20 \text{ cm}^3$. A comparison of the pulse height distribution between GHEISHA Monte Carlo and data for π at 200 MeV/c is shown in fig. 13.

To explain the peak at ~ 280 ADC-channels, we had to assume a muon contamination of the beam of 5% with a momentum resolution of 0.5%. The momentum resolution for pions has been quoted to be smaller than 0.1%. The width of the peak for non-interacting particles has been fitted by selecting a sensitive trigger time of the ADC's of 1 μsec . The signal for electrons of 200 MeV/c momentum would lie at ADC-channel ~ 500 .

4. FINAL REMARKS

We have seen a few examples of how to apply Monte Carlo techniques to the simulation and detection of processes in H.E.P. In these applications, Monte Carlo is a very powerful technique allowing the simulation of any kind of process, provided that the underlying theory is known. The full simulation of detectors needs Monte Carlo to account for all the incidences that a particle could have along the different compounds, as well as the inherent fluctuations to the multiple interactions in the development of showers.

In Monte Carlo programs, such as EGS and GHEISHA, phenomena without a clear theory or description behind, must be parametrized based on experimental data (so it is normally the case for the cross sections hadron-nucleus). Some parameters have to be tuned up in order to reproduce experimental distributions trying to balance the efficiency and accuracy of the program. Quantities, such as cross sections, might be changed inside the limits given by the experimental uncertainties in the tuning process.

In general there is some freedom in the programs allowing the tuning according to the experimental set up. Special care must be taken in situations where fine effects are important.

* * *

REFERENCES

- 1) Measurement and Analysis of Random Data. J.S. Bendat and A.G. Piersol. John Wiley & Sons, Inc. New York 1966.
- 2) Monte Carlo theory and practice. F. James. Rep. Prog. Phys., Vol. 43, 1980.
- 3) Introduction to Random Processes. E. Wong. Springer-Verlag, New York-Heidelberg-Berlin 1983.
- 4) Mathematical Software. Edited by John R. Rice. Academic Press, New York and London. 1971 (Pages 331 - 345).
- 5) Mathematical Methods for Digital Computers. A. Ralston and H.S. Wilf, Vol. II. John Wiley & Sons, Inc. New York-London-Sidney 1967 (chapter 12).
- 6) Random number generators. T.E. Hull and A.R. Dobell. SIAM Review 4, 230-254 (1962).
- 7) CERN program Libraries.
- 8) F.A. Berends, R. Kleiss and S. Jadach. Nuc. Phys. B202 (1982) 63-88.
- 9) F.A. Berends and R. Kleiss. Nuc. Phys. B177 (1981) 237-262.

- 10) F.A. Berends and R. Kleiss. Nuc. Phys. B178 (1981) 141-150.
- 11) A. Ali, DESY report 81-016 (1981).
- 12) T.D. Gottschalk. Lectures at the 19th International school of Elementary Particles Physics. Kupari-Dubrovnik, Yugoslavia, 1983. CALT-68-1075.
- 13) R.L. Ford and W.R. Nelson. SLAC, Standford, California. SLAC-210, UC-32, June 1978.
- 14) H. Fesefeldt., III Physikalisches Institut, RWTH Aachen, Fed. Rep. of Germany. PITHA-report, RWTH Aachen (To be published).
- 15) V.L. Highland, NIM 129 (1975) 497.
- 16) H. Fesefeldt et al. Nuc. Phys. B147 (1979) 317.

EXPERT SYSTEMS: AN OVERVIEW

Felisa Verdejo

Facultad de Informática, Universidad del País Vasco, Spain.

ABSTRACT

The purpose of this article is to introduce readers to the basic principles of rule-based expert systems. Four topics are discussed in subsequent sections: (1) Definition; (2) Structure of an expert system; (3) State of the art and (4) Impact and future research.

1. WHAT IS AN EXPERT SYSTEM?

The effort of many years of research in Artificial Intelligence (A.I.) has led to practical results. One of the most promising fields has come to be known as expert systems. An expert system is a computer program that solves problems by using specific domain knowledge-based reasoning. Expert systems have been developed mainly in areas such as medicine, geology, chemistry, and computers. They act as advisors on problems requiring judgmental knowledge to combine the relevant factors leading to a decision.

Characteristics that define an expert system are:

1. **Expertise:** an expert system performs tasks normally requiring significant human expertise in an interactive mode with the user.
2. **Symbolic processing:** in an expert system, knowledge is represented in symbolic form and manipulated by inference mechanisms involving uncertainty.
3. **Explanation:** an expert system is able to explain its way of reasoning (how and why it reaches conclusions).
4. **Flexibility:** an expert system acquires knowledge incrementally by increasing the knowledge base it becomes more expert.

Some of the well-known expert systems are:

- DENDRAL¹⁾ - deduces a chemical structure from a formula and its corresponding mass spectogram.
- MYCIN²⁾ - diagnoses patients with bacterial infections.
- PROSPECTOR³⁾ - aids in problems of mineral exploration to determine the most favorable drilling sites.
- XCON⁴⁾ - configures computer systems from orders, insuring that they are complete and specifying the spatial relationships among components.
- SACON⁵⁾ - selects an appropriate analysis strategy for studying a case with the MARC system (a computer program to simulate a mechanical structure behavior to various load conditions).
- DIPMETER ADVISOR⁶⁾ - interprets oil field logs to decide where to drill wells.

2. STRUCTURE OF AN EXPERT-SYSTEM

It is widely recognised that the source of a human expert's performance is the domain specific knowledge that he uses. Conventional programs encode knowledge as algorithms. A different approach is to express the task knowledge in a declarative form separated from the control component. Several schemas have been proposed, here we will mention two of them:

- . Predicate calculus. The information is represented as assertions and manipulated by theorem proving techniques. PROLOG, a general purpose computational language with an inference procedure based on the resolution principle is a well known example.

. Ruled-based systems. Domain knowledge is expressed in a uniform way by means of rules and separated from the procedures that will use it.

Most of the expert systems developed so far are rule based. The emphasis in this framework is not on the theoretical aspects but on its conceptual adequacy, applicability, and performance. In this paper we will focus on the rule-based approach.

2.1 Rules

Roughly speaking a rule is a conditional statement either relating facts or establishing the conditions to perform some actions.

One example of rule relating facts is the SACON rule outlined below

"IF the material composing the sub-structure is one of the metals,
AND the tolerable analysis error is between 5 and 30,
AND the non-dimensional stress is greater than 0,9,
AND the number of loading cycles to be applied is between 10 and 100
THEN fatigue is one of the stress behavior in the sub-structure."

A representative rule of the form "condition -> action," extracted from XCON follows:

"IF the current subtask is assigning devices to unibus modules
AND there is an unassigned dual port disk drive
AND the type of controller it requires is known
AND there are two of such controllers, neither of which has devices assigned to it
AND the number of devices which these controllers can support is known
THEN assign the disk drive to each controller
AND note each controller supports one device."

2.2 Knowledge base

A knowledge base is formed by the set of rules that synthesize the body of human expertise in the problem domain. Some rules would express causal interconnection among facts with a true value in the logical sense. Others would suggest what knowledge is relevant to a class of problems. Some other rules would represent likely relations observed among phenomena involving words such as "often" or "sometimes". These criteria learned in general by experience and practise, are called rules of thumb or heuristics in A.I. terminology. When used in reasoning about a problem, they are manipulated with a degree of confidence. Conclusions are reached by making conjectures and increasing their evidence. The final advice includes a measure of belief for the suggested interpretation.

2.3 Data base

The data base of an expert system contains the facts (known or inferred) about the current case being studied. The following are examples of facts:

The material of the substructure is one of the metals.

The tree has needles bundled together.

J. Smith has a stiff neck.

Starter turns engine slowly or not at all.

There is a type-A porphyry copper deposit (2.65)

The numerical value associated with the last statement is a measure of its degree of evidence.

Facts can be represented by a variety of data structures: arrays, strings, trees, or networks.

2.4 Inference procedure

The inference procedure tests the rules to determine which ones are satisfied by the descriptions of the data base. Depending on the interpretation of the rules, there are two basic modes of operation.

A) Backward-chaining. Rules have the form

IF<ANTECEDENT> THEN <CONSEQUENT>

A rule's antecedent has the general form C1 AND.....AND Cn where each Ci can be a disjunction of facts.

These rules are interpreted as follows:

To establish the goal <CONSEQUENT> look for evidence of <ANTECEDENT> in the data or knowledge base.

As a simple example, let us consider the following rules to identify trees:

1. IF tree is a conifer AND the needles are bundled together AND bundles have fewer than six needles THEN it is a pine.
2. IF tree has needles or scales like leaves THEN it is a conifer.
3. IF tree is a conifer AND needles are bundled together AND bundles have more than six needles THEN it is a larch.
4. IF tree is a larch AND the needles are three-sided THEN it is a western larch.
5. IF tree is a larch AND the needles are four-sided THEN it is a subalpine larch.
6. IF tree is a pine AND the needles are over seven inches long THEN it is a turkey pine.
7. IF tree is a pine AND the needles are shorter than 1½ inches THEN it is a foxtail pine.

With these rules, the hypotheses we can explore are:

Tree is a western larch

Tree is a subalpine larch

Tree is a turkey pine

Tree is a foxtail pine

Let us suppose the user wants to verify:

My tree is a western larch

With backward chaining mode, the interpreter attempts to conclude the goal by recursively establishing sub-goals until evidence is found. For each sub-goal the process splits into the following cases:

- Confirmed by pattern-matching within the database and therefore proved.
- There exist rules that permit to conclude it. These rules will be tried and the process continues.
- Rejected by pattern-matching or no rules could be applied. In both cases the current sub-goal fails.

In the case of our example, the goal follows from the premises of rule 4, which, therefore will be invoked.

```
-- larch
|
|
western larch-
|
| -- needles three sided
```

"Tree is a larch" and "needles three sided" are again hypotheses (sub-goals) to be determined. Subgoals are tried successively. In our example, to prove the hypothesis "tree is a larch", rule 3 might be selected.

```
-- conifer
|
|
larch - - - - - needles bundled together
|
| -- bundle more than six needles
```

At this stage rule 2 could be applied. The system would ask the user
"Has the tree needles or scales-like leaves?"

After receiving confirmation the process continues until there are no more subgoals to attempt. When they are proven, the goal is then said to be concluded.

Backward chaining strategy is goal oriented. It begins with a given hypothesis and looks for the conditions to be satisfied. The user is asked questions only if the system can no longer infer information from its knowledge base.

B) Forward chaining. Rules have one of the two forms:

IF <CONDITION> THEN <ACTION>

Example: IF starter turns engine slowly or not at all THEN perform headlight test

IF <PREMISE> THEN <CONCLUSION>

Example: IF there is voltage at the battery AND there is no voltage at the condenser THEN the wire between the battery and the condenser is broken.

and the interpreter executes as follows

> match the left-hand side (Situation/Premise) within the data base

> if the match is successful the rule might be selected

> to apply the rule means

- . to perform the action part possibly modifying the content of the data base (first form)
- . to add the consequent as a new fact. Actions in the second form are restricted to adding facts to the data base.

Forward chaining strategy is data-driven. It begins with the data provided by the user, selecting rules by pattern-matching and executing the corresponding actions.

Let us emphasize that systems based on rules following the pattern:

IF <ANTECEDENT> THEN <CONSEQUENT>

are deduction systems and can be interpreted either forward, or backward. Choosing a mode depends on the structure of the domain and the objectives pursued.

As we have seen, the interpreter matches the content of the data base with the rules. In the case that several rules are satisfied, a mechanism to decide which one should be applied is needed. Among the different criteria that have been used are the following:

- . Execute the first satisfied rule.
- . Express the way of selecting by rules (meta-rules).
- . Consider first the most/least used rule.
- . Select the rule with the larger number of facts (the more specific).
- . All the rules will be tried.

By now we have introduced the basic architecture of a rule based expert system, here is a summary:

1. A knowledge base of rules capturing the expertise of a domain.
2. A database of facts containing the information of the current case.
3. An inference procedure with:
 - a matching mechanism to compare the content of the database with the rules,
 - b. a mode of inference, backward or forward chaining,
 - c. a conflict resolution strategy to select among the satisfied rules.

The system performs by executing cycles formed by three steps:

- . Recognize (by matching) the candidate rules to apply,
- . Select a rule (following the conflict strategy),
- . Apply the rule

and continues until a solution is reached or there are no more rules to apply.

The separation of knowledge from control and the expression of knowledge by means of rules are the main features that make this framework useful. The first one enables us to use the same inference procedure for different domains; the second offers -but does not guarantee- an extensibility so that rules can be added/deleted without affecting the other components. This allows designers to develop large systems incrementally.

2.5 Explanation

There is an additional benefit to structuring knowledge in form of rules: the reasoning process becomes transparent and provides the system with an explanatory power. The user can ask questions about WHY a fact was or not considered, and HOW a conclusion was reached. The answers are based on the rules currently being applied. Following is an example from MYCIN:

System - Was penicillinase added to this blood culture?

User - WHY (do you ask this question?)

System - This will aid in determining whether organism-1 is a contaminant.

It has already been established that

- . The site of the culture-1 is blood
- . The gram stain of organism-1 is grampos

Therefore if penicillinase was added to this blood culture,

then is weakly suggestive evidence (.4) that organism-1 is a contaminant.

The required answer is extracted from the rule the system is working on

 -- penicillinase added

 |

 |

Organism-1 is a contaminant (.4) - - - - site is blood

 |

 |

 | -- gram-stain is grampos

"Why" questions are answered indicating the current conclusion to attempt and the facts already established.

The explanation facility is worth not only for the user but for the system builder. It provides him with an important debugging facility for the task of knowledge base development.

2.6 User's interface

An important issue in an interactive consultation system is the way in which the system communicates with the user. Expert systems typically don't use natural language but a variety of friendly menu-driven interfaces as well as other forms of entering information (graphics, signals,...).

2.7 Uncertainty

Expert systems have been developed mainly in areas where the human expertise conforms to a body of judgmental knowledge and conclusions can be reached by plausible reasoning with a certain degree of validity. Thus the previous architecture is usually augmented with a method to quantify evidence. This problem is under active research. At the moment most of the solutions built up came from the certainty, probability-based, or possibility models. The important problems they try to solve are the followings:

- . How to combine evidence from each rule's antecedents.
- . How to compute the strength of a conclusion.

How to treat the evidence of rules with interacting effects.

2.8 Summary

At this point, we can summarize the main components of a rule-based expert system:

- . The knowledge base - Domain dependent knowledge represented by rules.
- . The database - Facts about the problem at hand.
- . The inference procedure - For applying rules.
- . The explanation facility - To know about the reasoning process.
- . The user's interface - For interactive consultation.

3.STATE OF THE ART

Expert systems of practical applicability have been built and are at work. They perform well but are limited in an important number of ways. As experience grows, their strengths and weaknesses are better understood. A human expert is able not only to solve problems and explain what he does, but most important he can learn, reason on many levels, judge his own knowledge, restructuring and reorganizing it when appropriate. These skills are far from today's state of the art.

3.1 Current applications areas

Which are the criteria useful to decide when a rule-based expert system is adequate to solve a problem?. The major point concerns the central importance of knowledge. There must be an area where:

- . Human experts exist with a certain degree of agreement among them;
- . There are not effective mathematical models;
- . Knowledge is likely to be formalized and is not too difficult to express;
- . There is a natural way of decomposing expertise as rules with little interaction.

Other restrictions come from the state of the art:

- . Narrow domain of expertise, due to the lack of reliable methodologies and tools to create and maintain large knowledge bases;

- . Potential high benefits (intellectual or economical) due to the cost of building the knowledge base;
- . Lack of appropriate hardware at reasonable cost. Prices are dropping but machines allowing highly efficient symbol processing are still expensive.

Over the past decade a broad range of expert systems have been built to solve problems in different areas. The following list illustrates some of the applications:

- . Fault diagnosis
- . Medical diagnosis
- . Chemical data interpretation
- . Oil well log interpretation
- . Mineral exploration
- . Electronic troubleshooting
- . Signal interpretation
- . Speech understanding
- . Computer configuration
- . Computer aid instruction
- . Planning experiments
- . Electronic design.

Different attempts have been made to identify among the applications which are the generic tasks that would specify what an expert system does. A first characterization is to distinguish between analysis and synthesis. The analysis task consists of the identification or interpretation of data. Most of the systems are related to this task. What they do is classification problem solving, providing an answer from a preenumerated set of solutions. Problems of synthesis, such as planning or design, require that a solution be built up, usually verifying a number of constraints. These problems are difficult, but important research is going on and some significant results are being obtained. Often a domain task might be decomposed into a number of parts (either analytic or synthetic). An expert system for a practical application can combine both approaches.

3.2 Tools

An adequate software tool is a factor of great practical importance. Not only can it reduce the time of development but it can also help the designer to focus on the task-specific knowledge allowing him rapid prototyping to explore appropriate solutions.

Several possibilities are offered:

1. A selection of a rule-based framework such as EMYCIN or OPS, tools evolved from previous expert-systems and abstracted to being used in other domains. They usually provide
 - . A language to express rules
 - . Editing and debugging tools to create and maintain the knowledge base
 - . An inference mechanism
 - . An explanation module
 - . A consultation interface.

This option is worth in the cases where the structure of the new problem (generic tasks) fits the specification of one of the frameworks. For a detailed comparison of these tools see⁷.

2. There is a class of tools, (KEE, AGE, LOOPS..) more general than the previous one, that has no commitments to a particular form of rules or mode of inference. It allows to define different models. The price for this flexibility is more complexity. These languages (built on top of LISP) are suitable for advanced users or research work.
3. To use an artificial intelligence programming environment such as INTERLISP or POPLOG and write an expert-system beginning from the scratch. This way takes much more time.

In any case, as Davis points out, developing a substantial expert-system with real performance takes at least five man-years of effort, assuming the team has some background in A.I.

3.3 Implementation methodology

The process of building a serious expert-system is an empirical activity which requires two kinds of participants, human experts, who know about the problem, and so-called knowledge engineers who have to design the appropriate architecture as well as produce an initial knowledge base for the system. Transferring expertise from human to computers is recognized as a hard work, with an important investment of time and manpower. The main effort lies in the creation and maintenance of the knowledge base for the specific domain.

Systems already developed suggest the following construction stages:

Stage 1-producing a prototype. Main steps are:

- Problem definition: to identify the task, fix the role of the system and establish a common language between the domain expert and the knowledge engineer.
- Initial system design: to choose the knowledge representation and problem solving techniques.
- Knowledge acquisition: to extract knowledge from human experts and express it in a suitable form to be transferred to the system.
- Evaluation: to measure the behavior of the system in common situations.

Stage 2-Transferring the system to the operational environment.

- Extensive development of the knowledge base with special attention to cover ordinary and uncommon situations.
- Training the people to support the continuing improvement of the system.

Stage 3-Getting the system to work. Development, however, will never end due to changes in the problem domain.

The whole process can be characterized as one of progressive deepening. As the designers go on building a system, their understanding of the task grows, so do the knowledge base and performance of the system.

4. IMPACT AND FUTURE RESEARCH

Problems that resisted conventional software solutions have been successfully solved by expert-systems technology. This new approach has increased industrial expectation because of the high profits of replicating human expertise, specially in areas where there is a shortage of skilled personnel. For example, XCON (currently used by DEC corporation) has been saving the company 10 millions dollars a year.

Artificial Intelligence has been characterized as the study of ill-defined problems to be transformed into defined ones. The attempt to formalize a domain, clarifying progressively the principles on which decisions are made, has also significant side-

effects in educational and documentation uses. The knowledge base of MYCIN is now a good source of information for students. The new industry of "knowledge refinery", as Michie says, can be exploited to have enormous consequences on social life.

In the near future, expert-systems would be coupled with other already existing systems in operation, such as management information systems or mathematical packages to integrate different sources of knowledge.

Despite its accomplishments, the field is still in its infancy. Many problems need to be solved. Knowledge extraction remains a central bottleneck. Experiments in automatic knowledge acquisition range from intelligent editing to induction. Tools as EMYCIN offer an interactive way for transferring knowledge into its computer representation. METADENDRAL was able to build rules from data. Attention is being directed to other methods such as induction by examples or understanding textbooks, but the problem is hard to deal with and results difficult to be generalized.

We have only begun to understand some forms of reasoning. To make significant progress, basic research in fundamental issues such as causal models, common sense reasoning, and learning need to grow. The potentiality of the field is indeed vast.

* * *

REFERENCES

- 1) Lindsay, R. Buchanan, B. Feigenbaum, E. Leideberg, J. (80) - Applications of Artificial Intelligence for organic Chemistry: The DENDRAL project. New York. MC Graw Hill.
- 2) Buchanan, B. Shortliffe, E. (84) - Rule-based expert systems. The MYCIN experiment of the Stanford Heuristic Programming Project. Addison-Wesley.
- 3) Duda, R. Gasching, J. Hart, P. (79) - Model design in the Prospector consultant system. In "Expert-systems in the Micro-electronic Age", pp 153-167. Edinburgh University Press.
- 4) Mc Dermott, J. (80) - R1: an expert in the computer systems domain. Proc 1st National Conference Artificial Intelligence, pp 269-271.
- 5) Bennet, J. Engelmore, R. (79) - SACON: A knowledge-based consultant for structural analysis. Proc. Int. Conference on AI 79, pp 47-49.
- 6) Davis, R. Austin, Carlstrom, Pruchnik, Schneiderman, Gilreath (81) - The Dipmeter Advisor: Interpretation of Geological Signals. Proc. Int. Conference on AI 81.
- 7) Hayes-Roth, F. Waterman, D. Lenat, D. (83) - Building Expert Systems. Addison-Wesley.

ALGORITHMS FOR PARALLEL COMPUTERS

R. F. Churchhouse

University College, Cardiff, United Kingdom

ABSTRACT

Until relatively recently almost all the algorithms for use on computers had been designed on the (usually unstated) assumption that they were to be run on single processor, serial machines. With the introduction of vector processors, array processors and interconnected systems of mainframes, minis and micros, however, various forms of parallelism have become available. The advantage of parallelism is that it offers increased overall processing speed but it also raises some fundamental questions, including:

- (i) which, if any, of the existing "serial" algorithms can be adapted for use in the parallel mode?
- (ii) how close to optimal can such adapted algorithms be and, where relevant, what are the convergence criteria?
- (iii) how can we design new algorithms specifically for parallel systems?
- (iv) for multi-processor systems how can we handle the software aspects of the interprocessor communications?

Aspects of these questions illustrated by examples are considered in these lectures.

1. INTRODUCTION

Parallelism in a computer system means doing more than one thing at a time. The idea of parallelism in the context of computers can be traced back to a conversation of Charles Babbage in 1842 [1], as Professor Zacharov pointed out in his lectures to the 1982 CERN School of Computing [2]. As for its implementation: parallel handling of the bits of a word in arithmetic and logical operations was normal on most of the early electronic computers and parallel execution of I/O and computation was available on various general-purpose computers by about 1960. The overlapping of I/O and computation was built into the architecture of the machines and was achieved either by interrupts (hardware) or by polling (software).

For the purpose of these lectures "a parallel computer system" is defined to be a system of $p(>1)$ communicating processors or processing elements (P.E's), not necessarily identical, all of which can be in operation simultaneously, possibly under some form of central control. It may also be assumed that each P.E. has some private memory and has access to some common memory. This is (deliberately) a very vague definition: the "processors or processing elements", for example, may be anything from Cray XMPs to "AND gates"; in particular they may be simple, but specialised, devices such as those shown in Figure 1. The communication paths may be radial, nearest-neighbour, anything-to-anything or designed around some fancy geometry; the central control may dictate all the operations and enforce strict synchronization, or it may be almost non-existent.

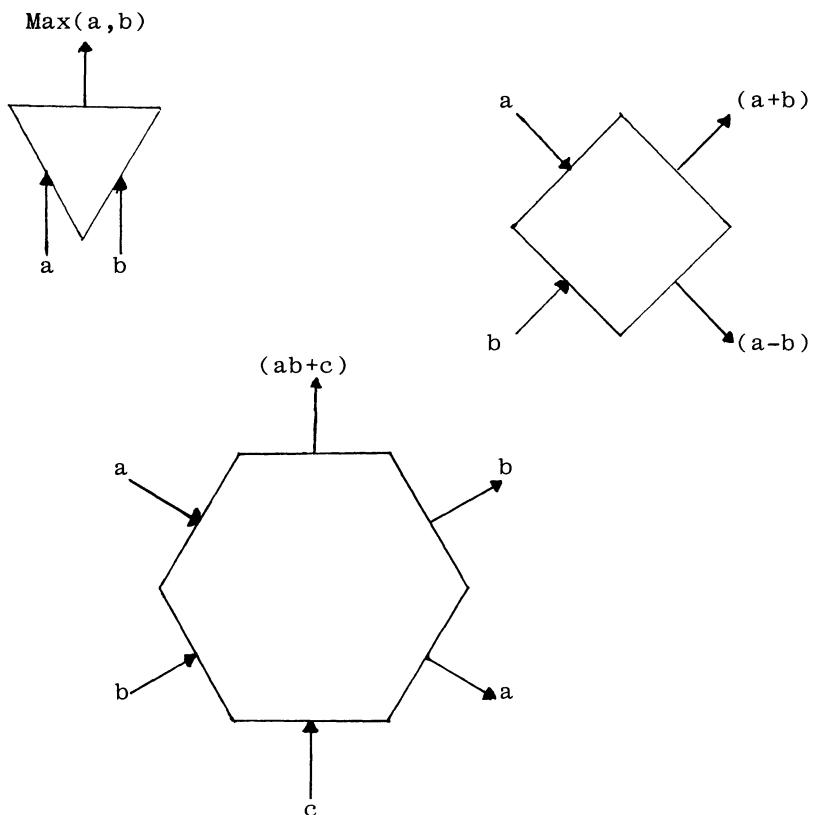


Figure 1.

If the P.E's all execute the same instructions in synchrony we have what Flynn [3] has called a Single-Instruction/Multiple-Data (SIMD) system. If the PE's can execute different instructions, whether synchronously or asynchronously, we have a Multiple-Instruction/Multiple-Data (MIMD) system. Examples of SIMD Systems include Illiac IV, the ICL-DAP, the Goodyear-NASA Massively Parallel Processor, the Cray-1 and the Cyber 205. We shall not be concerned with systems such as the Cray-1 or Cyber 205 or DAP in these lectures; they are (relatively) widely available and there are a very large number of papers on how to take advantage of their pipeline, vector or array facilities in a wide range of applications; see, for example [4, 5, 6].

MIMD Systems include multi-mini systems such as C.mmp, based on 16 PDP-11's, at Carnegie-Mellon University, the PACS-32 at Tsukuba University and the Neptune System at Loughborough University. We give more detail on these systems in Section 5.

A simple taxonomy of parallel computers which may be helpful, based on one given by Jordan [7], can be represented by the diagram below:

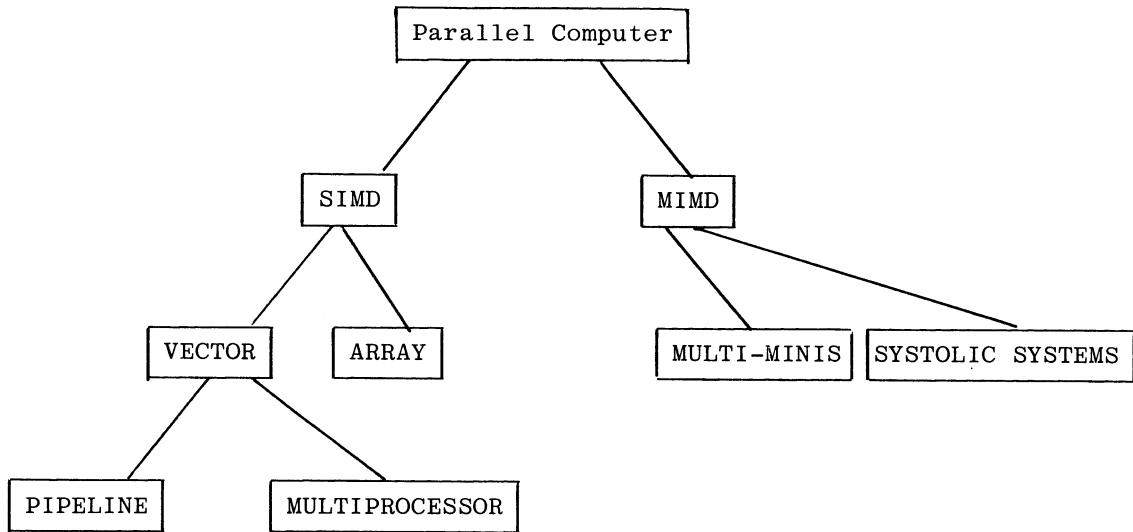


Figure 2.

The major reason for constructing parallel computer systems is, of course, that eventually the physical limitations imposed by the speed of light and switching times of elements will set a limit to the speed of computation of any serial computer, but this does not mean that the need for parallelism is based solely upon a desire to reduce the running-time of very long-running programs. There are other reasons, one is to increase availability and safety, whilst another arises in real-time environments where, say, a 10-millisecond response may be acceptable whereas a 100-millisecond response may be potentially disastrous.

There are then good reasons for trying to combine processors together and constructing algorithms to utilize them in parallel. Whether, however, we can increase the overall computation speed by an arbitrarily large factor by using a sufficiently large number of processors in parallel is an interesting question - the inter-processor communication times, which in the worse cases increase with p^2 , may eventually dominate the computation times which can at best only be expected to decrease proportionally to p^{-1} . Even for quite modest multi-mini systems the communication times may be very significant. Having raised this question we will not consider it further here, but it is important to stress that the communication times may impose very significant overheads in practice.

In these lectures we shall look at some examples of parallel algorithms and some implementations of parallel systems but first we introduce some basic concepts.

2. BASIC CONCEPTS

A program is considered to be composed of a number of processes, which it controls; the program itself may be controlled by an operating system. At any given time any particular process may be in execution in at most one processor, not necessarily the same processor at different times.

A parallel algorithm is an algorithm for the execution of a program which involves at least intermittent running of two or more processes on two or more processors simultaneously.

A synchronized parallel algorithm is a parallel algorithm which involves at least one process which is not permitted to enter a certain stage of its activity until another process has completed a certain stage of its own activity.

If T_1 is the time taken to run the 'best' serial algorithm for a particular program on a single processor and if T_p is the time taken by a parallel algorithm for the same program on a p-processor system then:

$$\text{the speed-up ratio is : } S_p = \frac{T_1}{T_p}$$

and the efficiency of the parallel algorithm is: $E_p = \frac{S_p}{p}$.

Sometimes we are interested in the speed-up and efficiency of a parallel algorithm compared to a serial version of the same algorithm, rather than the "best" algorithm. In such a case we might use the terms "relative speed-up" and "relative efficiency".

Example Suppose we wish to compute $xy + \log(x+y)$ using two PE's; if the computation of xy takes t_1 units of time and the computation of $\log(x+y)$ takes t_2 units, where an addition takes 1 unit, and where (as is very likely) $t_1 < t_2$, the computation is achieved in (t_2+1) time units by the synchronized parallel algorithm

<u>Time</u>	<u>Activity</u>
0	$x,y \rightarrow PE1; x,y \rightarrow PE2$
t_1	(xy) available from PE1; PE1 idles until input received from PE2.
t_2	$\log(x+y)$ sent from PE2 to PE1
(t_2+1)	$(xy)+\log(x+y)$ available from PE1

On a single PE the total time for this computation would be (t_1+t_2+1) units (ignoring storage times) so the speed-up ratio is $(t_1+t_2+1)/(t_2+1)$ in this case. Notice that this ratio must be less than 2, since we have assumed that $t_1 < t_2$, and hence the efficiency is less than 1. This result is typical: for a p processor system the speed-up factor will almost always be less than p, and the efficiency less than 1. There are, however, cases where a relative speed-up factor greater than p, i.e. a relative efficiency greater than 1, can be achieved (see Section 4.6).

An asynchronous parallel algorithm is a parallel algorithm with the following properties:

- (i) there is a set of global variables to which all processes have access;
- (ii) when a stage of a process has been completed the process first reads some global variables (possibly none) then, depending upon the values of these variables and the results just obtained from the last stage, the program modifies some global variables (possibly none) and then either activates its own next stage or terminates itself. In order to ensure logical correctness the operations on global variables may need to be programmed as critical sections [9].

Thus in asynchronous algorithms communications between processes are achieved by means of global variables. There is no explicit dependency between processes, as occurs in synchronized algorithms; asynchronous algorithms have the characteristic that they never wait for inputs but proceed according to the values of the global variables. Note, however, that some processes may have critical sections to which entry is blocked temporarily, as mentioned above [10]. Although the analogy isn't perfect we can think of synchrony as involving interrupts (hardware) and asynchrony as involving polling (software).

Apart from purely theoretical interest the main reason for the construction and analysis of asynchronous algorithms is that parallel algorithms are unlikely to succeed in keeping all P.E.'s equally busy and so the introduction of synchronization will inevitably lead to a slow-down in overall performance. Unfortunately a fully asynchronous parallel algorithm on several processors involving several global variables might well defy analysis, so that we would be unwise to use it in practice; it is, for instance, quite possible that the algorithm might converge with one assignment of processors and fail to converge with another assignment, even with identical initial inputs, as the experiment described in the next section illustrates.

3. SIMULATION OF THE NEWTON-RAPHSON ALGORITHM USING THREE P.E.'S

in (i) serial mode, (ii) synchronous parallel mode, (iii) asynchronous parallel mode.

In this simulation the Newton-Raphson algorithm, defined by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (3.1)$$

is used to solve $f(x) = 0$, from a given starting value x_0 , using three PEs which function as follows:

- (i) Given x , PE1 computes $f(x)$ in time t_1 units and sends it to PE3,
- (ii) Given y , PE2 computes $f'(y)$ in time t_2 units and sends it to PE3,
- (iii) Given a, b, c PE3 computes $d = a - b/c$; if $|d-a|$ exceeds a preset threshold the value of d is sent to PE1 and PE2, otherwise the value of d is

printed and the program halts. This all takes 1 unit of time.*

(*There is no loss of generality in this, if we are prepared to accept non-integral values of t_1 and t_2 ; for if $t_3 \neq 1$ we divide t_1, t_2, t_3 by t_3 and the conclusions are unaltered.

(i) Serial mode. Given x_n as input PE1 computes $f(x_n)$, after which PE2 computes $f'(x_n)$, after which PE3 computes x_{n+1} from (3.1), tests $|x_{n+1}-x_n|$ and takes appropriate action. The time taken for each such iteration is (t_1+t_2+1) units. If k such iterations are required the total time for completion will be $T_{SM}=k(t_1+t_2+1)$ units.

(ii) Synchronous parallel mode x_n is provided as input to both PE1 and PE2 simultaneously. When both have completed their tasks the values of x_n , $f(x_n)$ and $f'(x_n)$ are used by PE3 as in the serial case. The time taken for each such iteration is

$$T_{SP} = (\text{Max}(t_1, t_2) + 1) \text{ units.}$$

The number of iterations required will also be k so the total time to completion will be

$$T_{SP} = k(\text{Max}(t_1, t_2) + 1) \text{ units.}$$

(iii) Asynchronous parallel mode PE1 and PE2 begin computing as soon as a new value of the input variable is made available to them by PE3 and they are ready to receive it. PE3 computes a new value, using (3.1) in the form

$$x_{n+1} = x_n - f(x_i)/f'(x_j) * \quad (3.2)$$

as soon as either PE1 or PE2 provides a new input, and tests $|x_{n+1}-x_n|$ etc. as in the serial case.

The time for each iteration will be at most

$$(\text{Min}(t_1, t_2)+1) \text{ unit}$$

but it may be as low as 2 units (e.g. if PE1 and PE2 complete at consecutive time steps). In this case we cannot predict how many iterations will be required; it is unlikely to be less than k and may be significantly more - including, as we see below, infinity (i.e. the algorithm never terminates).

A program to carry out this simulation under these three modes for various sets of values of t_1 and t_2 for the function $f(x) = x^2 - 2$ with $x_0 = 1.0$ produced the following results; the threshold was $\frac{1}{2} \times 10^{-7}$ and $k=4$, in this case.

t_1	t_2	T_{SM}	T_{SP}	T_{AS}	S_S	S_A
1	5	28	24	20	1.17	1.40
2	5	32	24	21	1.33	1.52
3	5	36	24	38	1.50	0.95
4	5	40	24	31	1.67	1.29
5	5	44	24	30	1.83	1.47
6	5	48	28	∞	1.71	0

(T_{SM}, T_{SP}, T_{AS} are times to completion in serial, synchronous parallel, asynchronous parallel modes; S_S and S_A are the relative speed-up factors for the synchronous and asynchronous modes).

*(the values of $(n-i)$ and $(n-j)$ are bounded, with explicit bounds, but that needn't concern us here).

In the case $t_1=6$, $t_2=5$ the algorithm fails to converge in the asynchronous parallel case; the system falls into an approximate cycle of length 36 time units during which six new approximations to the solution are obtained, none of them correct to 1 d.p. The reason for this is that the convergence of the N-R algorithm depends critically upon the value taken at each stage for $f(x_n)$ but depends to a much lesser extent on the value taken for $f'(x_n)$, unless this happens to be very small. Use of $f(x_n)$ rather than $f(x_{n+1})$ can cause the correction term to have the wrong sign whereas use of $f'(x_n)$ instead of $f'(x_{n+1})$ will usually leave the algorithm convergent but linearly, not quadratically. By taking $t_1 > t_2$ it is inevitable that we will sometimes use a value for $f(x_n)$ which has the wrong sign, so destroying the convergence.

The main purpose of the example above is to illustrate an important point; an algorithm which is quite satisfactory on a serial machine, or even on a synchronized parallel system, may fail completely on an asynchronous system. Even if the asynchronous version ultimately terminates satisfactorily it may take longer than a synchronized, or even a serial, version. Whilst there are serial algorithms, such as the Gauss-Seidel and Jacobi methods for solving systems of linear equations, which can be run successfully on asynchronous parallel systems (see, for example the paper by Baudet [11]), in general this will not be the case. We conclude therefore that, in the majority of cases, algorithms for parallel computers, whether synchronous or not, will need to be specially designed and not merely be adaptations of existing serial algorithms.

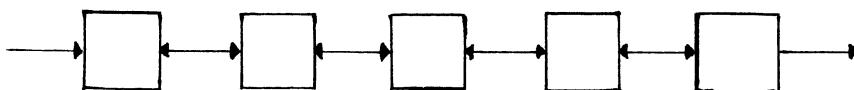
4. SOME COMMUNICATION GEOMETRIES AND PARALLEL ALGORITHMS

In this section two algorithms which have been adapted to MIMD systems are described. The first is a simple sorting algorithm which, in its parallel form, is naturally of synchronous type; the second is a generalisation of the Binary Search algorithm and will be given both in synchronous and asynchronous form.

As a preliminary it is convenient at this point to explain the nature of some communication geometries which occur frequently both in the literature, and in practice.

4.1 Linear communication

In these systems the P.E's are assumed to be arranged (from a geometrical, if not from a physical, point of view) in a line, viz:



For $2 \leq k \leq (n-1)$ $PE(k)$ can communicate with $PE(k-1)$ and $PE(k+1)$. $PE(1)$ can communicate with $PE(2)$ and $PE(n)$ can communicate with $PE(n-1)$.

If $PE(1)$ and $PE(n)$ can communicate with each other we have

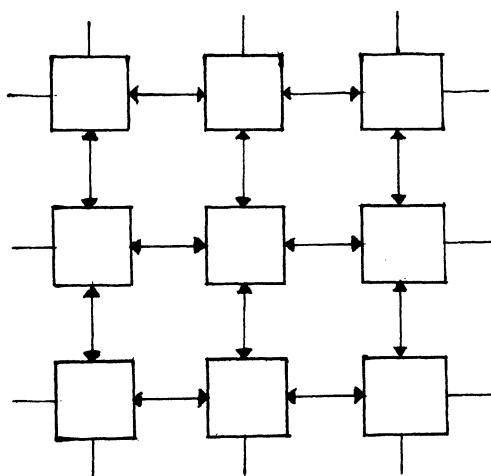
Ring communication otherwise we have One-dimensional Linear Communication.

Simple though such a system is it suffices for many applications and its suitability for shift registers and pipelines is apparent.

4.2 Nearest-neighbours, two-dimensional arrays

In this system the PE's are envisaged as lying at the points to a two-dimensional rectangular lattice. Each row and each column has one-dimensional linear communication so that each P.E is connected to 4, 3 or 2 other PE's depending on whether it lies at an internal lattice point, a boundary lattice point or a corner lattice point.

Thus a 3x3 array can be represented



As in the linear-case, "end-around" communication 'horizontally' and/or 'vertically' is sometimes included, giving rise to 'cylindrical' or 'toroidal' communication geometries.

4.3 Two-dimensional hexagonal geometry

Two-dimensional space can be completely filled by (i) identical rectangles, (ii) identical equilateral triangles, (iii) identical hexagons. Kung [12] has utilised hexagonal symmetry to produce an elegant parallel processing system for multiplying two-dimensional matrices, as we shall see in Section 4.7 where this system will also be described.

4.4. The Odd-Even Transposition Sort

This is one of the simplest methods of sorting; on a uniprocessor it has the advantage of being easy to program and the disadvantage of being very inefficient, since the number of operations required to sort N items is $O(N^2)$, so that it is unlikely to be used if N is more than a few thousand. In a parallel mode however it is quite interesting because the sort can be easily performed by a collection of very simple PE's; with N such PE's the sorting time is reduced to $O(N)$ units so that for special

purpose applications, where a few thousand items must be sorted quickly, a special-purpose device could be economic.

For simplicity we suppose that N , the number of items to be sorted, is even and write $N=2m$. The odd-even transposition sort proceeds as follows:

Odd phase: For $i=1$ to m compare the items in positions $(2i-1)$ and $(2i)$ and interchange them if they are in the wrong order.

Even phase: For $i=1$ to $(m-1)$ compare the items in positions $(2i)$ and $(2i+1)$ and interchange them if they are in the wrong order.

Repeat both phases m times.

The list will then be in sorted order. The total number of comparisons/interchanges will be $m(2m-1)$ so, if we define a comparison/interchange to take unit time the total time required will be about $\frac{1}{2}N^2$ units.

If we connect together N PE's in a linear array, without end-around communication, as in 4.1, where $PE(k)$, when activated, compares the item in its store with the item in the store of $PE(k+1)$ and interchanges them if they are out of order then the synchronized parallel odd-even transposition sort on N PE's proceeds as follows:

Odd phase: Activate all odd-numbered PE's in parallel.

Even phase: Activate all even-numbered PE's, except $PE(N)$, in parallel.

Repeat both phases m times

The list will then be in sorted order.

The total time required, ignoring any synchronization overheads, is N units. The speed-up factor is therefore about $\frac{1}{2}N$ and the efficiency about $\frac{1}{2}$.

The parallel system described above is a simple example of what Kung calls "a systolic architecture" [13]. "A systolic system consists of a set of interconnected cells, each capable of performing some simple operation.... Information in a systolic system flows between cells in a pipelined fashion and communication with the outside world occurs only at the "boundary cells". For example, in a systolic array, only those cells on the array boundaries may be I/O ports for the system". In a systolic system data and intermediate results flow between the PE's in a rhythmic fashion, analogous to the flow of blood within the body, hence the name.

Many standard sorting methods have been re-designed for parallel systems and sorting times of orders $O(N)$, $O(N^{\frac{1}{2}})$, $O(\log^2 N)$ and $O(\log N)$ achieved, the faster methods usually requiring more PE's and/or more complex intercommunication and control. For a more sophisticated version of the odd-even sort see a recent paper by Wong and Ito [14], for other methods see [15].

4.5 Binary Search

This algorithm is perhaps the simplest both for finding an item in an ordered list and (under the name "Bisection Method") for finding a zero of a

continuous function to a specified precision, given that it is known to lie in some interval. The analysis is essentially the same for both cases; we shall describe it for the latter case. The algorithm is certain to succeed and although there are more efficient algorithms this one has the advantage that we can predict very accurately how many iterations will be required.

We suppose that we have a continuous function, $f(x)$ and that we have two points x_0 and x_1 such that $f(x_0) < 0$ and $f(x_1) > 0$; we therefore know that $f(x)$ has a zero in the interval $\langle x_0, x_1 \rangle$ of length $|x_1 - x_0|$. The Bisection Algorithm in serial form proceeds as follows:

- (1) Evaluate $f\left(\frac{x_0+x_1}{2}\right)$; if the value is zero go to the print routine, if it is negative replace x_0 by $\frac{1}{2}(x_0+x_1)$, if it is positive replace x_1 by $\frac{1}{2}(x_0+x_1)$;
- (2) If $|x_0 - x_1|$ is sufficiently small go to the print routine, otherwise go back to (1).

The interval in which the zero lies is halved in length at each iteration so the number of iterations required to improve the accuracy from k to $(k+n)$ decimal places will be

$$\lceil n \log_2 10 \rceil$$

(i.e. the integer greater than or equal to $\frac{10n}{3}$).

If we have a parallel system composed of p P.E's the obvious parallel equivalent of the Bisection Algorithm involves dividing the interval containing the zero into $(p+1)$ sub-intervals of equal length at each iteration. The function, $f(x)$, is evaluated at the $(p+1)$ division points simultaneously and when all the evaluations have been completed the choice of the new interval is made. Synchronization is clearly essential for this algorithm and there will be time penalties associated both with the synchronization and with choice of the new interval. If we ignore these overheads the number of iterations required to improve the accuracy from k to $(k+n)$ decimal places will be

$$\lceil n \log_{(p+1)} 10 \rceil$$

- but because of the overheads it will be more than this. The speed-up factor of this synchronized parallel algorithm is therefore at most

$$\frac{\lceil n \log_2 10 \rceil}{\lceil n \log_{(p+1)} 10 \rceil} \doteq \frac{\log_2 10}{\log_{(p+1)} 10} = \log_2(p+1)$$

and the efficiency is

$$< \frac{\log_2(p+1)}{p}$$

which decreases as p increases, so this algorithm is likely to give better value for money, so to speak, when p is fairly small. For $p=4$, for example, it gives an efficiency of at most 0.57.

The Binary Search/Bisection Algorithm has received considerable attention in the literature and variations on the fully synchronized parallel version described have been published.

4.6 Binary Search : Asynchronous version using two P.E's.

If we have two P.E's we can execute the synchronized binary search algorithm of section 4.5, achieving a speed-up factor, ignoring overheads due to synchronization, etc., of $\log_2 3 (\approx 1.59)$.

The search can, however, also be carried out asynchronously. Kung [10] describes the algorithm in the two P.E case in detail and a simplified version of this is given below.

We begin by assuming that we have two points A and D, a continuous function $f(x)$ and that $f(A)f(D) < 0$. Let $|D-A|=L$ and let θ be the positive real number defined by

$$\theta^2 + \theta - 1 = 0$$

(so that θ is $\frac{1}{2}(\sqrt{5}-1) = 0.618\dots$)

In the asynchronous algorithm to be described we shall, at any instant, know that the zero lies in a certain interval and be awaiting the evaluation of $f(x)$ at two points; when either of these evaluations is complete the process either terminates or a new evaluation is begun at a point which is determined by:

- (i) which of the two evaluations was completed,
- (ii) which of two "states" (defined below) the system was in during the evaluation,
- (iii) the value of $f(x)$ at the point of evaluation.

The two states are defined with reference to the interval in which the zero is known to lie and the two points at which the evaluation is being performed. As above, let the interval in which the zero is known to lie be $\langle A, D \rangle$ and be of length L. Define points B,C inside $\langle A, D \rangle$ by

$$B = A + \theta^2 L, C = A + \theta L.$$

From the definition of θ it follows that $|CD| = \theta^2 L$ and $|BD| = \theta L$; less obviously, $|BC| = \theta^3 L$. The values of $f(x)$ at the points A and D are, of course known.

In state $S_1(L)$: we are awaiting the evaluation of $f(x)$ at the point B and at some point, E, outside $\langle A, D \rangle$.

In state $S_2(L)$: we are awaiting the evaluation of $f(x)$ at the points B,C.

The evaluations at the two points are carried out asynchronously by the two P.E's. On completion of an evaluation a P.E either terminates the algorithm or updates 5 global variables: (i) the system state, (ii) the end-points of the interval in which the zero is now known to lie, (iii) the values of $f(x)$ at these two end-points.

The appropriate action to be taken when an evaluation is complete and

$|f(x)|$ is not yet sufficiently small to terminate the algorithm can best be described by a table. For convenience we suppose that $f(A) < 0$ and $f(D) > 0$ and indicate the evaluation just completed by a + or - sign and the evaluation still in progress by a ? sign. By $S_1(I)$, $S_2(I)$ we indicate that the zero lies in an interval of length I .

Case	Present State	Sign of $f(x)$ at points				New State	New Interval
	$S_1(L)$	E	A	B	D		
1		?	-	-	+	$S_1(\theta L)$	$\langle B, D \rangle$
2		?	-	+	+	$S_1(\theta^2 L)$	$\langle A, B \rangle$
3		-	-	?	+	$S_2(L)$	$\langle A, D \rangle$
4		+	-	?	+	(Implies more than one zero : ignore)	
	$S_2(L)$	A	B	C	D		
5		-	-	?	+	$S_2(\theta L)$	$\langle B, D \rangle$
6		-	+	?	+	$S_1(\theta^2 L)$	$\langle A, B \rangle$
7		-	?	-	+	$S_1(\theta^2 L)$	$\langle C, D \rangle$
8		-	?	+	+	$S_2(\theta L)$	$\langle A, C \rangle$

Thus, in Case 6, for example: the evaluation at B is completed first and $f(B) > 0$ so the zero is now known to lie in the interval $\langle AB \rangle$ of length $\theta^2 L$ ($\approx 0.382L$); the value of $f(C)$ is still awaited and since C lies outside $\langle AB \rangle$ we are now in state $S_1(\theta^2 L)$. The PE which evaluated $f(B)$ now proceeds to evaluate $f(x)$ at the point P in $\langle AB \rangle$ defined by

$$P = A + \theta^2 |AB| = A + \theta^4 L$$

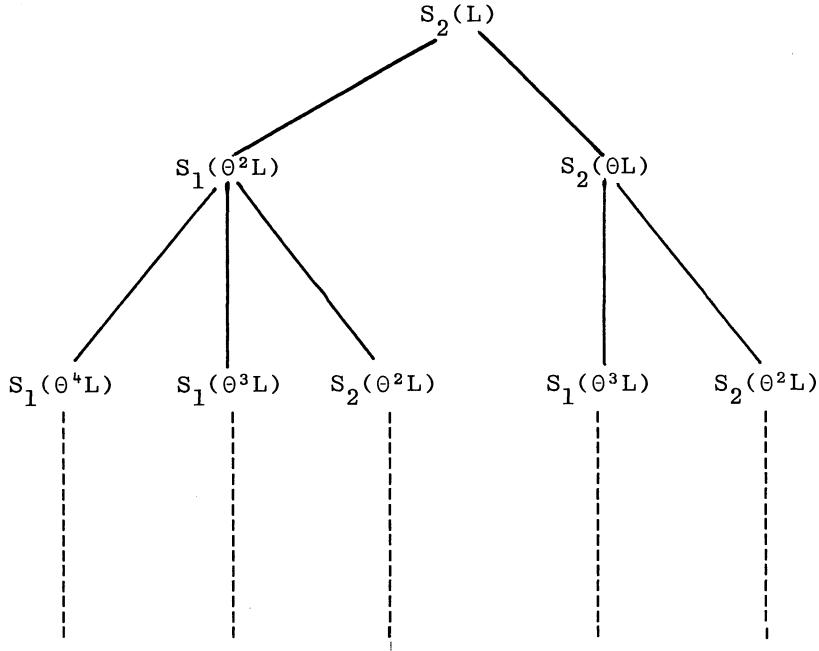
as required by state $S_1(\theta^2 L)$, and updates the five global variables accordingly. (We ignore the fact that the value of $f(C)$ is irrelevant, in this version of the algorithm, in Case 6: more complex versions exploit this fact, see [10]).

The algorithm is certain to converge since the interval in which the zero is known to lie is reduced in length by a factor of θ (0.618...) in three of the seven cases, by a factor of $\theta^2(0.382...)$ in three of the remaining cases and is left unchanged in Case 3. Since, however, Case 3 represents a transition from $S_1(L)$ to $S_2(L)$ it cannot occur again immediately so that a reduction of interval length must follow after the next evaluation even in this case.

The speed at which the algorithm converges depends upon which particular path through the tree (which is derived from the table above and takes $S_2(L)$ as initial state) corresponds to the sequence of functional evaluations. Kung (op.cit) shows that the speed-up, compared to single P.E binary search lies in the interval

$$\langle 2 \log_2(\theta^{-1}), 4 \log_2(\theta^{-1}) \rangle \text{i.e. } \langle 1.388, 2.777 \rangle$$

and that the asynchronised two P.E. algorithm is certainly faster than the



synchronised version if the overheads due to synchronization exceed 14% and could be as much as about 1.5 times faster even if the synchronization overheads are negligible.

This algorithm illustrates how complex the analysis of an asynchronous algorithm can be even for a two P.E system. As the number of P.E's increases so do the number of possible state-transitions and so too the complexity of the analysis also increases. Note, too, that even if all the P.E's are identical the computation times for $f(x)$ at two points, x_1 , and x_2 say, may be very different e.g. if $f(x)$ is a slowly converging series, rather than a polynomial. On the other hand the synchronization overheads will also increase as the number of P.E's in a synchronized system is increased and it is not obvious what number of P.E's is optimal for the two types of algorithm, particularly if all the P.E's are identical (if they are not the synchronized algorithm may lose rather heavily, the overall speed being dependent on the slowest P.E).

4.7 Matrix multiplication by Systolic Arrays with Hexagonal Symmetry

Kung [12] has shown how an array based upon identical, very simple, P.E's can be used to multiply two two-dimensional matrices. The system is elegant and illustrates very nicely the systolic array concept.

The basic P.E can be described geometrically as a hexagon, with 3 input-faces and 3 output-faces as shown in Figure 3.

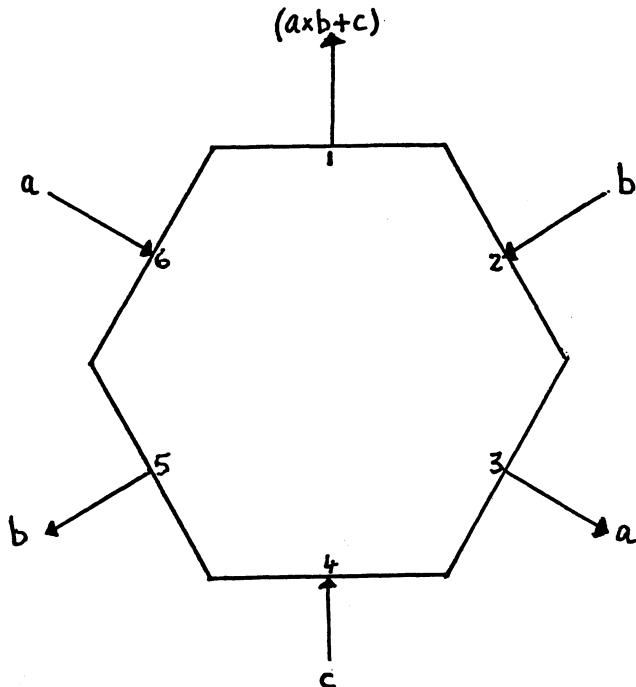


Figure 3.

The function of the P.E is very simple, viz: Inputs a , b and c arrive at faces 6, 2 and 4 and emerge as outputs a , b and $(axb+c)$ at faces 3, 5 and 1 respectively one clock-pulse later.

Suppose now that we wish to multiply two 2×2 matrices

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

producing

$$C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

we may do this by building a system of 7 P.E's as shown in Figure 4.

The various data streams at time zero are shown in the figure. What happens at each subsequent clock-pulse is shown below:

Time

Activity

1 $C_{11} = a_{11}b_{11}$

2 $C_{11} = C_{11} + a_{12}b_{21}; C_{21} = a_{21}b_{11}; C_{12} = a_{11}b_{12}$

3 $C_{21} = C_{21} + a_{22}b_{21}, C_{12} = C_{12} + a_{12}b_{22}; C_{22} = a_{21}b_{12}$

Thus with 7 P.E's we multiply two 2×2 matrices in 4 clock pulses. If the matrices are 3×3 we add another ring of 12 P.E's "outside" the outer 6 shown

in Figure 4. For $n \times n$ matrices we require

$$1 + 6 + 12 + \dots + 6(n-1) = 3n^2 - 3n + 1$$

P.E's and the computation is completed in $O(n)$ clock pulses. On a uni-processor multiplication of two $n \times n$ matrices normally takes $O(n^3)$ operations (we assume that the matrices are "full" and ignore the possibility of using one of the ingenious, but complex, algorithms such as Strassen's [16].

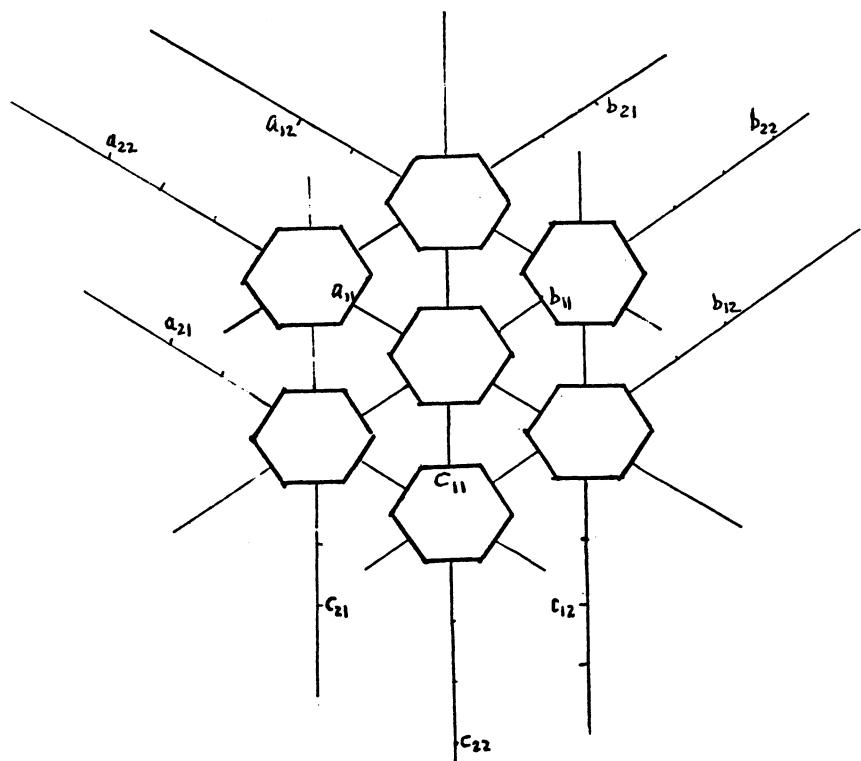


Figure 4.

Kung gives variations on this method for Band Matrices and applies the same system to the Computation of Discrete Fourier Transforms.

It is by no means inconceivable that applications exist where fast multiplication of two two-dimensional matrices of moderate order (eg $n=20$) is required in which case the building of a special systolic array based on less than $3n^2$ (1141 for $n=20$) identical simple P.E's of the type above might provide a satisfactory solution.

5. SOME IMPLEMENTATIONS OF PARALLEL ALGORITHMS ON MIMD SYSTEMS

The practical implementation of a parallel algorithm on an MIMD system is never easy and may be very difficult. Even when the system is homogeneous, i.e. when all the processors are identical, the standard operating system and compilers for a single processor will not be suitable for the multi-processor system so that until operating systems and compilers for languages which include parallelism become widely available, specially written programs at the machine-code level are likely to be required.

Despite these difficulties, however, MIMD systems have been constructed and a variety of algorithms written for them and tested to see how they perform in practice. The Department of Computer Science at Carnegie-Mellon University has played a major role in this respect and their reports,[e.g.13, 17] form a most interesting and valuable part of the research literature, which should be studied by anyone interested in the subject. Here we can only pick out a few examples.

In [17] Oleinick reports on the implementation and evaluation of parallel algorithms for (i) root-finding, and (ii) speech-recognition on the Carnegie-Mellon C.mmp system. The C.mmp system is described in detail in [18]. For our purposes it is sufficient to know that, at the time when [18] was written, C.mmp was a system of 16 processors that shared a common memory of 2.5 Megabytes. The 16 processors were completely asynchronous PDP-11s; 5 were PDP-11/20's and the remaining 11 were PDP-11/40's. Connection of the processors to the memory was via a 16 x 16 crosspoint switch, permitting up to 16 memory transactions simultaneously. I/O devices, unlike memory, were associated with specific processors so that use of a device by another processor required inter-processor communication. A general-purpose multiprogramming operating system, called Hydra [19], was written for the system; it was organized as a set of re-entrant procedures that could be executed by any of the processors; several processors could execute the same procedure simultaneously, this concurrency being accomplished by the use of *locks* around critical sections of the operating system.

The root-finding algorithm used by Oleinick was the 'natural' generalization of the binary-search algorithm when n P.E's are available, i.e. subdivision of the interval of uncertainty into (n+1) equal-length subintervals. This is not the optimal subdivision (see [17], for the general case, and section 4.6 above for n=2) but it is not far off and has the merit of simplicity. The problem solved can be stated:

"Given a random number h lying in $\langle 0,1 \rangle$ and a fixed small number ϵ find the value of x such that

$$|F(x)-h| = \left| \frac{1}{\sqrt{2\pi}} \int_x^{\infty} \exp(-\frac{1}{2}t^2) dt - h \right| < \epsilon$$

Computation of the integral was done by means of the series

$$x + \frac{x^3}{3.5} + \frac{x^5}{3.5.7} + \frac{x^7}{3.5.7.9} + \dots$$

for $x < 2.32$

and by means of the continued fraction

$$\frac{1}{x+} \quad \frac{1}{x+} \quad \frac{2}{x+} \quad \frac{3}{x+} \quad \dots$$

for $x > 2.32$.

Computation times varied for different values of x . At each stage the n processors computed the values of $f(x)$ at the n division points. When all values were available the last processor to complete its computation tested for completion of the algorithm and either terminated or worked out the new sub-interval, "woke up" the "sleeping" processors and so activated the next stage. The algorithm is therefore of the synchronous type and the overall speed is bounded above by that of the slowest processor and subjected to the overhead of the synchronization and other delays.

Oleinick compared the actual speed-up achieved on n processors ($n=2,3,\dots,9$) with the best-possible theoretical speed-up [$\log_2(n+1)$], found that the gap between expected and observed performance increased as n increased (e.g. 3 processors performed at about 80% of the expected level whereas 9 processors performed at about 70%) and analysed the causes of the degradation of performance. Seven factors were found:

- (1) Synchronization: synchronization mechanisms covering a wide range of sophistication were tried, producing significantly different results as the granularity increased
- (2) Calculation of F(x): the time required for this varied with x , up to a factor of 3.4
- (3) Memory Contention: the average cycle length varied by a factor of up to 2.8
- (4) Bottleneck of Scheduling processes in the Operating System: again a factor of up to 2.8 in delay time was observed
- (5) Speed of the Processors: in the non-homogeneous C.mmp individual processor speeds varied by a factor of up to 1.6
- (6) I/O Devices and Interrupts: could degrade computation of $F(x)$ by a factor of up to 1.3
- (7) Variations in memory speed: the 16 memory banks associated with the PE's were not identical (some were core, some semi-conductor), this caused a further degradation by a factor of up to 1.07.

Speech recognition systems offer the opportunity of exploiting parallelism but their complex control structures can impose a large synchronization overhead [20]. Oleinick [17] describes the use of the Carnegie-Mellon speech recognition system HARPY on C.mmp. HARPY can

recognise phrases and sentences from many speakers based on a finite vocabulary. A serial algorithm already existed and this was refined in various ways to produce four versions of a parallel algorithm, the fourth having some asynchronous sections. Using 8 processors speed-up was improved from 2.94 in the first version of the parallel algorithm to 4.29 in the fourth version. Significant improvements resulted from (i) balancing the work-load across all the processors, (ii) arranging for two sub-tasks to run asynchronously rather than, as hitherto, synchronously.

In another experiment C.mmp based on 4 processors recognised 15 trial utterances in 46 seconds, compared to 49 seconds on the (more powerful) uniprocessor DEC KL10. Using 7 processors C.mmp completed the recognition in 33 seconds. (On 1 processor C.mmp required 144 seconds).

This example illustrates both the effort that may be required to construct satisfactory parallel algorithms, particularly when the synchronization overheads are likely to be large, and the rewards that may be achieved.

5.2 PACS Hoshino and others at the University of Tsukuba have described a 32-element parallel microprocessor array for scientific calculations called PACS-32 [21] and have given details of its performance on a variety of problems including aerodynamics, Monte Carlo, nuclear reactor calculations and the solution of partial and ordinary differential equations.

PACS-32 is a MIMD system with both synchronous and asynchronous modes, built around 32 processing units, together with a control unit and a host computer as well as I/O devices and communication lines (see Figure 5).

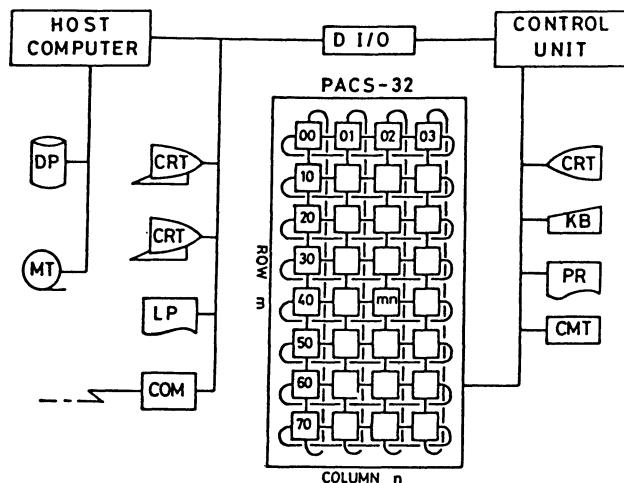


Figure 5. Configuration of PACS-32 system. (CRT = cathode ray tube, KB = keyboard, PR = printer, CMT = cassette magnetic tape, DI/O = digital input/output lines, PU = processing unit, LP = line printer, COM = communication interface, DP = disk pack memory, and MT = magnetic tape.) (From Hoshino et al. [21])

The 32 processing units (P.U) are represented geometrically as an 8×4 array, as shown in Figure 5 ; physically they are arranged in a $2 \times 2 \times 8$ rectangular box. Each P.U, including those on the left-and right-hand sides, top and bottom, is connected to its 4 nearest neighbours (as indicated in Figure 5). Topologically therefore the 32 P.U's are arranged as a torus.

Each PU contains several elements, including a micro-processing unit (MPU), an arithmetic processing unit (APU), as well as local, result and communication memories. The MPU'S are Motorola MC6800 (8-bit) micro-processors and the APU's are Advanced Micro Device's Am9511's (16 and 32-bit arithmetic). (See Figure 6).

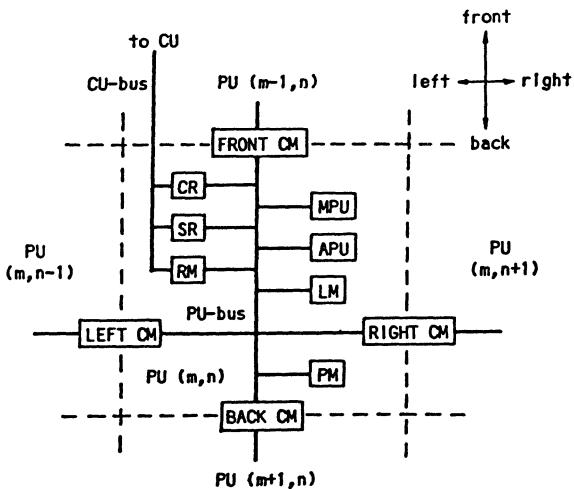


Figure 6. Configuration of processing unit (MPU = microprocessing unit, APU = arithmetic processing unit, LM = local memory, RM = result memory, FRONT CM, BACK CM, LEFT CM, and RIGHT CM = communication memories in front, back, left, and right, respectively, CR = control register, SR = status register, PU (m, n) = processing unit with coordinate (m, n) , CU = control unit). (From Hoshino et al. [21])

Each PU has access to three banks of memory: Local Memory (5 kbytes) for local data and programs; Result Memory (1 kbyte) shared between the PU and Control Unit and used for communication between them; Communication Memory (2kbytes) shared between neighbouring PU's. In order to avoid access conflict in the Communication Memory all PU's are synchronized by the system clock; this clock signal originates in the control unit and is fed to all PU's but for the odd-numbered PU's it is fed 180° out of phase and hence odd- and even-numbered PU's access the Communication Memories in

alternate half-cycles of the clock. Since the neighbours of PU(m,n) are PU($m,n\pm 1$) and PU($m,n\mp 1$) and since $m+n$ determines whether a PU is "odd" or "even" the four neighbours of PU(m,n) are necessarily of opposite parity to PU(m,n) itself; hence conflicts in the Communication Memory are avoided.

The host computer is a Texas Instruments 990 Model 20. It is used to compile/assemble the source program, load the object code into the CU and PU, initiate parallel tasks, transfer and receive the data to and from the CU, and output the results.

Recognising that different applications would require different degrees of inter-processor communication and synchronization the designers of PACS-32 have provided facilities to allow considerable flexibility in these areas. Data transfer is available in two modes: *communication* and *non-communication*; in the latter case there is no data transfer among PU's; Monte-Carlo simulation for independent particles utilizes this mode. The *communication mode* is further subdivided into *synchronous* and *asynchronous* modes. In the *synchronous mode* all PU's are synchronized by the statement CALL SYNC. After the PU's have been synchronized, data are transferred to the Communication Memory, the transfers being organized to avoid memory contention. The *asynchronous mode* is itself subdivided into *conditional* and *unconditional* modes. In the conditional asynchronous mode data transfer occurs only if the data have been updated; in the unconditional asynchronous mode data are transferred regardless of whether they have been updated or not.

Programs for PACS-32 are usually written in FORTRAN for the serial part of the job and in SPLM, a high-level language for parallel-processing based on SIMPL [22], for the parallel parts of the job.

The performance of PACS-32 was evaluated over six major problems and the efficiency of the system, defined as the percentage of the time that the PU's were busy, worked out. The efficiency ranged from 78% to 99%. The author's final paragraph is worth quoting *verbatim*:

"Computer scientists and engineers used to assume that overhead due to inter-processor communication would prohibit the practical application of a PACS-type processor array. Our experience has shown that this assumption can no longer be made."

6. PARALLEL LANGUAGES

None of the widely-used early programming languages, such as FORTRAN, ALGOL or COBOL, were designed to cope with parallel algorithms. With the introduction of multiprocessor systems however the possibility of parallel programming was raised and a number of proposals for handling concurrency, as in Concurrent Pascal [24], or parallelism (e.g. ALGOL 68 [25]) appeared. More recently ADA, which caters for parallelism by means of "Tasks" has been

introduced [26], but few people have so far had the opportunity of using it.

For the vector and array processors the manufacturers (CRAY, ICL etc.) have provided modified versions of FORTRAN which enable the programmers to exploit the particular features of these various machines, some of these versions of FORTRAN are very machine-specific, which is perhaps not surprising. Portability of programs from one such machine to another seems a long way off. For a good summary of the situation c. 1981 see [27].

For the MIMD systems the situation is even less satisfactory. Most of the systems built so far are essentially "one-off" and their creators have generally provided ad hoc solutions to the programming problem, often by means of additional features in FORTRAN. Fairly typical are those provided at Loughborough University in the UK for their NEPTUNE system [28]. NEPTUNE is a parallel system based upon four Texas Instruments 990/10 minis running under the DX10 Operating System. Pseudo-FORTRAN syntactic constructs have been provided to enable users to

- (i) create and terminate parallel paths
- (ii) state which data is shared between paths
- (iii) ensure synchronization where required.

Thus for example the construct:

```
$DOPAR 100 I = N1,N2,N3  
      (code)  
100 $PAREND
```

causes the generation of $(N_2 - N_1 + 1)/N_3$ paths each with a unique value of the loop index I. This is used for the generation and termination of paths with identical code.

On the other hand to generate paths with different code a construct of the form

```
$FORK 101,102,103;150  
101  (code 1)  
      TO TO 150  
102  (code 2)  
      GO TO 150  
103  (code 3)  
150  $JOIN
```

is used.

Synchronization is provided by the user specifying a list of "resources" which may only be owned by one processor at any time. Such a resource is claimed and subsequently released by the construct

```
$ENTER (name 1)  
$EXIT  (name 1)
```

If any other processor tries to claim this resource in the interim it is refused and the processor idles until the resource is available. It is clear that a deadlock situation could arise; the onus of preventing this is left to the programmer.

REFERENCES

- (1) Menabrea, L.F, "Notions sur la machine analytique de M.Charles Babbage", Bibliotheque Universelle de Geneve, Serie 3, Tome 4, (1842), 352-376.
- (2) Zacharov, V, "Parallelism and Array Processing", Proc. 1982 CERN School of Computing, 66-121.
- (3) Flynn, M, "Some computer organisations and their effectiveness", I.E.E.E. Trans. Comp C-21,9 (1972), 948-960.
- (4) Rodrigue, G (Ed), "Parallel Computations", Academic Press (1982).
- (5) Evans, D.J. (Ed), "Parallel Processing Systems, An Advanced Course", Cambridge University Press (1982).
- (6) Howlett, J. et al, "DAP in action", ICL Technical Journal, May 1983, 330-344.
- (7) Jordan, T.L, "A guide to Parallel Computation and some Cray-1 Experience" in (4), 1-50.
- (8) Barlow, R.H. and Evans, D.J, "Parallel Algorithms for the Iterative Solution to Linear Systems", Comp.J, 25 (1982), 56-60.
- (9) Dijkstra, E.W. "Cooperating Sequential Processes" in "Programming Languages" (F.Gennys, Ed), Academic Press (1968), 43-112.
- (10) Kung, H.T, "Synchronized and Asynchronous Algorithms for Multiprocessors" in "New Directions and Recent Results in Algorithms and Complexity" (J.F.Traub, Ed), Academic Press (1976), 153-200.
- (11) Baudet, G.M, "Asynchronous Iterative Methods for Multiprocessors", J.A.C.M, 25 (1978), 226-244.
- (12) Kung, H.T, "The Structure of Parallel Algorithms", Advances in Computers, 19 (1970), Academic Press, 65-112.
- (13) Kung, H.T, "Why Systolic Architectures?" Carnegie-Mellon Dept. of Computer Science Report: CMU-CS-81-148 (1981).
- (14) Wong, F.S. and Ito, M.R, "Parallel Sorting on a Re-circulating Systolic Sorter", Comp.J, 27 (1984), 260-269.
- (15) Thompson, C.D and Kung, H.T, "Sorting on a Mesh-Connected Parallel Computer", CACM, 20 (1977), 263-271.
- (16) Strassen, V, "Gaussian Elimination is not Optimal", Numerische Mathematik, 13 (1969), 354-356.
- (17) Oleinick, P.N, "The Implementation and Evaluation of Parallel Algorithms on C.mmp", Carnegie-Mellon University, Dept. of Computer Science Report, CMU-CS-77-151 (1978).
- (18) Wulf, W.A and Bell, C.G, "C.mmp - A Multi-Mini Processor", AFIPS Proc. FJCC 1972, Vol. 41, 765-777.
- (19) Wulf, W.A. et al, "HYDRA: The Kernel of a Multiprocessor Operating System", C.A.C.M. 17 (1974), 337-345.
- (20) Lesser, V.R, "Parallel Processing in Speech Understanding Systems", Speech Recognition (1975), 481-499.
- (21) Hoshino, T et al, "PACS: A parallel microprocessor array for scientific calculations", ACM Transactions on Computer Systems, 1, (1983), 195-221.
- (22) Basili, V.R and Turner, A.J, "SIMPL-T: A Structured Programming Language", Univ. of Maryland, Computer Science Center Report CN-14.1 (1975).
- (23) Hoare, C.A.R., "Communicating Sequential Processes", C.A.C.M, 21 (1977), 666-677.
- (24) Brinch Hansen, P: "The programming language Concurrent Pascal", IEEE Trans. Software Eng 1 (1975), 199-207.
- (25) van Wijngaarden, A (Ed): "Report on the Algorithmic language ALGOL 68", Numer.Math 14 (1969), 79-218.
- (26) Shumate, Ken; "Understanding ADA", Harper and Row (New York), 1984.
- (27) Hockney, R.W. and Jesshope, C.R: "Parallel Computers", Adam Hilger Ltd. (Bristol), 1981.
- (28) Barlow, R.H. et al: "The NEPTUNE parallel processing system", Dept. of Computer Studies, Loughborough University, UK (1981).

TRENDS IN SUPERCOMPUTERS AND COMPUTATIONAL PHYSICS

T. Bloch

Centre de Calcul Vectoriel pour la Recherche, Ecole Polytechnique, Palaiseau

1. Introduction

Investigation and evaluation of complex non-linear systems is frequently only possible using numerical modelling. This - almost experimental - computational science is increasing in scope and importance every day thanks to the combination of improved computational methods and the continuous development of faster and faster "supercomputers".

Application fields which were pioneered in the 1950's and the 1960's have today entered the production stage; e.g. structural safety calculations, global circulation models for meteorological forecasting, aerodynamics' design, oil reservoir simulation. Use of supercomputers and numerical modelling permit considerable global economies in agriculture and transport due to improved weather forecasting and, in engineering applications, shorter development cycles, lower design costs and lighter end designs are achieved.

Today, scientists using numerical models explore the basic mechanisms of semiconductors, apply global circulation models to climatic and oceanographic problems, probe into the behaviour of galaxies and try to verify basic theories of matter, such as Quantum Chromo Dynamics by simulating the constitution of elementary particles. Chemists, crystallographers and molecular dynamics researchers develop models for chemical reactions, formation of crystals and try to deduce the chemical properties of molecules as a function of the shapes of their states. Chaotic systems are studied extensively in turbulence (combustion included) and the design of the next generation of controlled fusion devices relies heavily on computational physics.

Thus, many complex systems which cannot be studied analytically with present methods and which are not open to experimentation, are studied on supercomputers improving our basic scientific understanding. As a result the scientific and economic importance of supercomputers and the advanced use of them is now clearly perceived by all major industrial countries. More than 100 CRAY and Cyber 205 computers are installed world-wide with Europe accounting for about one third and Japan about 10%. Although Japan thus seems to lag slightly behind on supercomputer applications, the trend is to catch up very rapidly now that both Hitachi and Fujitsu have proven machines available (S-810 and VP-200) and NEC is bringing out the SX-2 by the middle of 1985.

In the United States there are more supercomputer installations than anywhere else, but it has now been generally accepted - thanks for a large part to K. Wilson - that the access for the university research community has not been as good as in Europe where an appreciable proportion of supercomputers are installed in generally accessible regional or inter-university computer centres: one in Holland, two in the United Kingdom, one in France, one in Italy and four or five in Germany. A major effort is now under way in the United States to distribute the access to supercomputers much more

widely; this programme is handled through the National Science Foundation which, already in 1985, is funded to set up about three new supercomputer centres.

2. Examples taken from the early history of computational physics

The starting point for using numerical methods in cases where analytical ones are no longer possible can perhaps be traced back to the work of W.F.Sheppard concerning the finite difference methods. This method was used by L.F. Richardson from 1910 to make a dam computation after which he went on to conceive the first global circulation model imagining 64,000 people in a huge amphitheatre, each advancing a time-step at a time based on the information from the neighbours under the baton of a conductor. R. Courant, R. Friedrichs and H. Lewy (1928) brought the theoretical foundation that partial differential equations are indeed approximated by the discretized equations under certain stability criteria.

Around 1950, John von Neumann and a number of collaborators attacked new problems with the first digital computers (ENIAC in 1946, then MANIAC) and started to realize the programme of computational physics which he had published in 1946 together with H.H. Goldstine:

- develop stability criteria for time-step and grid size for finite difference methods to solve parabolic partial differential equations.
- make shock-wave calculations using smoothing techniques with boundary layers with at least the size of the grid.
- code the first barytropic (no temperature variation along isobars) general circulation model with a grid of 15 x 18 over the continental United States (with J.G. Charney, R. Fjörtoft and J. Smagorinski).
- make hydrodynamic instabilities and neutron cascade calculations with E. Fermi and S. Ulam in Los Alamos on MANIAC. On this occasion, S. Ulam discovered the Monte Carlo method.

The discovery of solitons is a classic example of a new insight into physics found by computational physics: in about 1955, E. Fermi, J. Pasta and S. Ulam studied the motion of a chain of masses coupled by weakly non-linear springs. (They were trying to develop a theory explaining the finite thermal conductivity of solids.) Long wavelength disturbances of this system resulted, as expected, in a mix of shorter wavelength modes. However, after a finite time, the system was unexpectedly seen to revert to (almost) the initial state (without equi-partition of the energy). In the early 1960's, M. Kruskal and N. Zabusky found that the problem studied could be approximately reduced to the Korteweg-de Vries equation which is a well known hydrodynamics equation describing shallow water waves. Numerical studies of the solutions to this equation lead to the discovery that its complicated solutions could be classified as a set of pulse-like waves propagating at constant velocity with an unchanged shape, called solitons. A fertile period of analytic proofs of many new properties of a variety of non-linear partial differential equations used in physics then followed.

3. Historical notes from the development of recent supercomputers

Several definitions of supercomputers exist; from the fast(est) computer(s) of a given period to the only computers which, already on the day of their first delivery, are not fast enough.

3.1 The "pure FORTRAN" supercomputers up to 1969

From about 1956 to 1969, it was the reign of the FORTRAN-machine in the field of the fastest computers: UNIVAC, Ferranti, IBM and CDC installed computers such as the LARK, the 7090, the 6600, the STRETCH, the ATLAS, the 360-91 and the 7600. During this period the speed of computers increased by a factor of two every two or three years, without significantly affecting the price and without the user having to restructure his programs. (In practice, of course, reliability problems and the painful evolution of the first multiprogrammed, and later, time-sharing, operating systems left much to be desired in the service provided.)

The first delivery of the CDC 7600 in 1969 marked the end of the reign of the FORTRAN machine in supercomputing; it was about 50 times faster than the IBM 7090, its senior by 10 years. Its basic circuits were about 10 times faster, and another factor of five has been achieved by internal improvements in architecture, allowing more things to go on in parallel both within the CPU itself, and also between the CPU and the stored information: multiple memory banks, instruction buffers and preloading of instructions, many more registers (programmable and otherwise) in the CPU, multiple and pipelined execution units (and the logic to keep track of what was going on where), etc...

From 1969 on, two major factors became dominant in determining which supercomputer could be developed:

- a) the basic technology no longer improved rapidly (the CRAY X-MP2, first delivered in 1983, is based on circuits only about three times faster than those of the CDC 7600) and the benefits even from these improvements became difficult to realise fully because the dominant factor in high-speed design has moved away from the switching speed of the active devices to the propagation delays in the interconnecting conductors.
- b) higher speeds with the same technology can be obtained if one gives up the "simulation", vis-à-vis the user, of the basic "von Neumann" architecture of the CPU: fetch an instruction, decode it (being prepared for anything), fetch the operands (from anywhere), execute the operation, store the result anywhere and start all over again for the next instruction.

The result is that new and faster "vector" computers are now available but the user must be sollicited much more. He must collaborate in order to organize the execution of his program better, he must tell the computer clearly through the program how the access to operands is organized when there is regularity and he must avoid IFs where he can and try to organise the computation in long DO-loops. Ultimately, he must rethink the problem, the method and the coding to suit the internal organisation of the computer...

The next step, already true for the CRAY X-MP and soon to be imposed by the CRAY-2, the ETA-10 and the Fujitsu VP-400 will be that the user must split his

problem into two, four or eight "pieces" which can execute concurrently on independent processors with a common main memory.

3.2 The "Vector" supercomputers from 1974 to 1985

Two pioneering supercomputers with this new vector architecture were conceived from 1966 onwards and first delivered in 1973/1974, the CDC STAR-100 and the Texas Instruments ASC (Advanced Scientific Computer). They had rich instructions sets including explicit vector instructions operating in a memory-to-memory fashion and multiple "pipelines" to execute them. However, they already clearly demonstrated the potential problems of these architectures:

- high peak vector speeds on long vectors contrasting only too sharply with the scalar speeds achieved. The execution time of any codes other than trivial ones, would often be "scalar" or "short vector" dominated.
- high cost of software; for the manufacturer to develop a complete operating and a vectorizing FORTRAN for a very small number of machines and for the user to convert his program to take advantage of the vector instructions.
- long (and costly) development cycle for the hardware resulting in use of seemingly out-dated technology in the delivered machines.

None of these two computers were commercial successes, both were sold to two or three sites with a few more used internally in Control Data or Texas Instruments data centres. The major reasons were rather poor reliability due to the massive amount of circuits and mediocre user acceptance due to the considerable conversion and rethinking effort necessary on each program before it would even execute as fast as it used to on a CDC 7600! (For well behaved codes, programs would, eventually, execute much faster but the investment of up to several man-years on each big program could only pay off in very specific cases.)

These computers did however pave the way for the acceptance of the CRAY-1 which was first installed in 1976 and of which about 100 are now installed, or on order (including its immediate successor, the CRAY X-MP2). Also the CDC Cyber 205, based on the STAR-100 architecture but using high-speed modern technology and first installed in 1981, has been widely accepted (about 40 are installed or on order). It remains that the CRAY-1 was the computer which made the concept of "vector supercomputing" acceptable by solving the main problems encountered till then:

- high scalar speed so that even the most "resistant" CDC 7600 program would normally execute faster on a CRAY-1 without any conversion effort.
- excellent hardware reliability.
- simple software approach allowing good user acceptance: batch operating system with only disks (and later tape units) and FORTRAN. Front-end communication protocol defined to allow a general purpose computer to feed the CRAY-1 with jobs, and files from tapes and disks and, in general, provide the user services such as time-sharing, networking, database systems, etc.
- good speed gains on relatively short vectors by using a vector register design rather than memory-to-memory design for the vector instructions.

For a superficial comparison of the characteristics of today's vector supercomputers including the latest arrivals from Fujitsu and Hitachi as well as the yet undelivered CRAY X-MP4, NEC SX-2, CRAY, ETA-10 (see Table 1).

3.3 The Lock Step Multiprocessors from the ILLIAC IV to the Goodyear MPP

Since the proposal of Slotnick et al in the early 1960's to construct the ILLIAC IV a few large computers have been constructed with the following "processor array" architecture:

- many identical execution processors each with their own local memory.
- one instruction decoding processor which directs the "army" of processors that all have to do the same instruction in the same cycle - but on different data (the possibility that some of the execution processors do nothing always exists).
- some interconnection scheme permitting data to move from one processor to another.

The major motivation for this sort of architecture is one of technological cost: it should be relatively economical to build a computer with the potential of doing many calculations per second based on the replication of a large number of identical small processors. However, the limited flexibility of the design, in particular the segmentation of the data into blocks per processor, makes it difficult to adapt complicated applications efficiently to such architectures, and if a non-infinitesimal portion of code only executes on one processor when the "normal" speed is based on 64, 4096 or even 16384 processors, the overall effect of this section of code on the speed quickly becomes dominating.

In spite of excellent algorithm and code development work done by UK scientists around the ICL DAP, it currently looks as if this sort of architecture will not be successful in the near future in the general purpose supercomputer market, but it seems a very serious candidate for signal processing applications; The Massively Parallel Processor (MPP) built for NASA by Goodyear to be used for image processing is, basically, a modernised and scaled-up version of the DAP.

3.4 Conclusions

To keep improving the speed of the new supercomputers, designers must accept a larger and larger gap between the peak speeds obtainable on a "well" adapted, highly regular and vectorized problem and the speed at which a "messy" part of a program runs. This gap varies from a factor ten on a CRAY-1, up to several thousands if we talk about a multiprocessor with so many processors, of which only one is in use at a given time, in critical parts of the code. The net result is that the execution speed of application programs get more and more sensitive to those parts which cannot be vectorized or executed in parallel. Thus the "scalar speed" starts to influence heavily the real useful speed of supercomputers, except in the case of rare dedicated applications adapted to benefit from the maximum speed of a given architecture.

4. Some Applications and their Ambitions

4.1 Meteorological forecasting

The dynamic behaviour of the atmosphere is characterized by non-linear phenomena and a situation where the initial state cannot really be known with a reasonable precision. The general circulation models used in meteorological forecasting and in climate studies model the evolution of the atmosphere using very coarse approximations at times (the shape and height of mountains, average albedo coefficients, etc.) and ignoring effects which are well known to be important but which are too complex to be included yet: coupling with the detailed surface state of the oceans and atmospheric chemistry for example.

A five-day forecast is today produced routinely by several centres around the world. Better models and better observational data from satellites should allow better medium range forecasts - maybe as far as 10 days within the next decade. Such improvements will also require much faster computers: the mesh size covering the earth for these calculations will have to be decreased significantly from the present 1° mesh and the time-step must be decreased (currently about 10 or 15 minutes). Present estimates are that a 10 GFLOPS computer will soon be necessary if these ambitions are to be realized.

4.2 Molecular Dynamics

This field covers the simulation of the statistical and dynamic behaviour of complex molecular systems: liquids, liquid mixtures and solutions, polymers and biological macro-molecules. Such systems contain thousands or tens of thousands of atoms with the order of about hundred interacting neighbours to each atom. Detailed quantum mechanical treatment is excluded from the start in such problems (the state of the art in quantum mechanics on today's supercomputers allows the treatment by ab initio calculations of molecules with a few tens of atoms, some of which can be metals...).

As an example, in a simulation of liquid water a system size of 200 particles gives rise to 20,000 molecular pairs (of which half are within interactive range). The forces on each atom are calculated for each dynamic step (typically 10^{-15} seconds for Newton's equations), 300,000 force components. Such a system needs about 10^4 time steps to evaluate its physical properties. A supercomputer in the CRAY-1 range can do this calculation in half an hour.

Interesting problems dealt with today concern aqueous solutions of simple proteins. They require more time-steps (because the movements of sections of macro-molecules are slow) and are much more complicated - more water molecules are needed to surround the macro-molecule and the forces are more complex because of the rotation around bonds in the macro-molecule. Typically, 30 hours of supercomputer time is needed here. In this field the amount of calculations is estimated to go up by three order of magnitude when one wants to treat large polymers, nucleic acids or organic systems - and another three orders of magnitude if one wants to study reactions.

4.3 Theoretical Physics

Problems in theoretical physics have been using a very large fraction of available supercomputer time over the last few years for the modelling of the Quantum Chromo Dynamics (QCD) theory of strong interactions of hadrons (by gluon exchanges between quarks). These calculations are done on lattices and are highly regular and well suited for getting the maximum performance out of present-day supercomputers. But they are constantly hampered by the limitations in CPU speed and memory size which prevent them from reaching a sufficiently fine discretization both in space-time and in the physics phenomena. Furthermore certain effects have to be deliberately ignored in the models (like everywhere else in computational physics) and whereas the calculation of the quark-antiquark potential is accessible to these methods the determination of the masses of the hadrons is not yet possible. The problems treated today use a space time-mesh inside a four-dimensional box to study the wave function of a hadron. It is presently estimated that a considerable improvement in resolution is necessary in order to achieve results with negligible systematic errors. The computing time required goes up with roughly the 5th power of the resolution, meaning that 100 hours of CRAY-1 time used for a study need to become 10 years of computation to determine accurately the hadronic mass spectrum. Still, KEK, the Japanese High Energy Physics Laboratory in Tsukuba will install a Hitachi S-819/20 supercomputer in Japan in June 1985 primarily for use by theoretical physicists.

4.4 Computational Aerodynamics

The central problem is the problem of aeroplane design: what are the flow patterns for various possible configurations of a future aeroplane with its engines at different speeds. This problem is governed by the Navier-Stokes equations which are too complex to envisage their complete solution on present day computers for configurations of practical interest. In order to treat the problem in practice one simplifies the geometry - two dimensional cross-section of a wing only - or one uses more or less complete and complex approximations to the Navier-Stokes equations such as (in order of increasing complexity): the potential flow equations, the Euler equations, the Reynold averaged approximation and the large eddy approximation. Today's largest supercomputers permit the modelling of a complete aircraft (500,000 mesh points) with the 3-D Euler equations in practice (few hours of computation).

In order to make realistic simulations of separated or unsteady flow where viscosity effects play a key role, it is, however, necessary to use better approximations than the Euler equations. The use of a Reynold averaged approximation on a complete engine-airplane configuration is estimated to require a computer 30 times a CRAY-1 (1 GFLOPS constant processing speed, not peak!) with at least 250 Mwords of main memory.

5. Conclusions and Comments about the Future

Each new progress in speed (still at roughly constant price...) of supercomputers seem to permit new advances in computational physics and applications: established

fields make progress in the complexity and realism of the problems they can investigate, new fields become amenable to modelling and numerical investigation and economically-inspired shifts from traditional methods towards computer treatment occur: aerodynamics, structural calculations, production of animated films, oil reservoir simulation, electrical distribution networks, design of semiconductors and VLSI, etc.

Although an improvement of a factor ten in computing speed seems marginal in some three-dimensional research applications where, at best, it allows a decrease of a factor two in grid size (the time-step size must be adjusted as well to keep numerical stability) the impact of computer speed improvement is reinforced by considerable improvement in computational methods from the model simplifications chosen through the basic numerical algorithms developed and on to the actual coding techniques. The progress that has been made in computational linear algebra is spectacular and without the algorithms available today for Fast Fourier Transforms, many applications would not be possible. In a specific example studied, the number of arithmetical operations required in order to solve a particular 3-dimensional partial differential equation decreased by four orders of magnitude in the period 1960 to 1980 by the evolution from finite difference methods with successive overrelaxation to finite element methods with multigrid computations - an improvement ten times bigger than the speed-up from the IBM 7090 to the CRAY-1 over the same period.

For the moment the lowest speed of a supercomputer improves only slightly - at best a factor of two has been achieved since the CRAY-1 of 1976 but it seems possible for the supercomputer designers to keep on considerably improving the hardware peak speeds - the 100 MFLOPS of the late 1970's is now becoming 1 GFLOPS and is expected to reach 10 GFLOPS over the next few years. To benefit from these speed improvements, the user must, thus, find algorithms and coding techniques which allow more and more vectorization (due to pipelined hardware) and explicit parallelism (due to multi-processor architectures). The overall data structures used are also becoming very critical (due to limitations in primary memory speeds and perhaps more importantly, due to severe limitations today in disk speeds).

An important trend in the design of supercomputers today is to make them with much larger memories: the CRAY-X-MP now goes as high as 16 MWords; the Fujitsu VP-200, the Hitachi S-810/20 and the NEC SX-2 come with 64 Mwords (32-bits) and the CRAY-2 processor machine with 256 MWords (64 bits). The ETA GF-10, announced for 1986 will also provide up to 256 MWords for its eight processors.

Thus the immediate future looks bright and interesting for those engaged in computational physics, but there is a lot of hard work to be done in order to develop and adapt algorithms to these very unbalanced architectures where peak speeds and lowest speeds now differ by about two orders of magnitude and where the interpretation of results - and subsequent improvement of the models used - require substantial work in the "post processing" of numerical model output, graphics (animated system evolving in time) in particular.

6. Acknowledgements

I would like to acknowledge the help of the users of the Centre de Calcul Vectoriel pour la Recherche and to Marie Farge in particular for material concerning the history of computational science and information about present applications.

Bibliography

On architecture (mostly) two standard books will serve as an introduction.

- 1) R.W. Hockney, C.R. Jesshope, Parallel Computers: Architecture, Programming and Algorithms. Adam Hilger Ltd., Bristol 1981 (very good coverage of the field)
- 2) K. Kwang, Briggs, A. Fayé, Computer Architecture and Parallel Processing, McGraw Hill 1984 (good level of detail and includes descriptions of some experimental systems (S-1) and of data flow computers - ignores the ICL DAP completely).

On algorithms and applications (mostly) three recent Conference Proceedings will give a good start on the field:

- 3) High Speed Computation. Proceedings of the NATO Advanced Workshop on High Speed Computation held at Jülich, Federal Republic of Germany, June 20-22, 1983. Janusc S. Kowalik (Editor) Springer 1984.
- 4) Proceedings of the International Conference on Parallel Computing, Freie Universität Berlin, 26-28 September 1983. Editors: M. Feilmeier, G. Joubert, U. Schendel. North Holland 1984.
- 5) Proceedings of the Conference on Vector and Parallel Processors in Computational Science II (VAPP II), Oxford, 28-31 August 1984. To be published by North Holland.

TABLE 1

Computer	Year of 1st Customer	Recent Vector Supercomputers				Style of Vector Computation
		Number of machines	CPU Cycle time & peak rate	Memory technology and size	Secondary semiconductor memory	
CRAY-1	1976	approx.70 (inc.CRAY-1M)	12.5 ns (160 MFLOPS)	Bipolar, now SRAM 4 Mwords (64 bits) 5.1 Gbits/sec.	Yes Yes	1 1
CDC CYBER-205	1981	approx.30	20 ns (800 MFLOPS)	Bipolar, 16 Mwords (32 bits) 25.6 Gbits/sec.	No	1
CRAY-X-MP2	1983	approx.10	9.5 ns (420 MFLOPS)	Bipolar, 4 Mwords (64 bits) 41 Gbits/sec.	Yes	2
Fujitsu VP-200*	1983	approx. 5 (inc. VP-100)	15 ns (533 MFLOPS)	SRAM, 64 Mwords (32 bits) 34 Gbits/sec.	No	1
Hitachi S-810	1983	approx. 4	14 ns (670 MFLOPS)	SRAM, 64 Mwords (32 bits) 34 Gbits/sec.	Yes	1
CRAY X-MP4	1985	-	9.5 ns (840 MFLOPS)	Bipolar, 8 Mwords (64 bits)	Yes	4
CRAY-2	1985	-	4 ns (1 GFLOPS)	DRAM, 256 Mwords (64 bits) 64 Gbit/sec.	No	4
NEC-SX-2	1985	-	6 ns (1.3 GFLOPS)	SRAM, 64 Mwords (32 bits) 88 Gbits/sec.	Yes	1
Reduced instruction set scalar architecture, IBM floating point format register to register vector instructions, 4 specialized sets of 4 pipes each, 10240 vector registers						

* A VP-400 with twice the performance has been announced in January 1985.

SUPERCOMPUTING AND RELATED NATIONAL PROJECTS IN JAPAN

Kenichi Miura
Supercomputer Planning Department
Mainframe Division
FUJITSU Limited, Kawasaki, Japan

Abstract

Japanese supercomputer development activities in the industry and research projects are outlined. Architecture, technology, software, and applications of Fujitsu's Vector Processor Systems are described as an example of Japanese supercomputers. Applications of supercomputers to high energy physics are also discussed.

1. INTRODUCTION

In recent years, demands for solving large scale scientific and engineering problems have grown enormously. Most of the computations in these problems involve vast amount of floating point operations on array data, and they can be handled very efficiently if suitable architecture is employed to exploit the inherent parallelism in the problems. Pipelining and parallel processing are typical techniques. In this lecture note, let us just define supercomputer as computer which can achieve more than one order of magnitude performance improvement over large mainframe computers for the kind of problems stated above. Since array data are often called vectors, vector processor and supercomputer (or vector processing and supercomputing) will be used interchangeably throughout this lecture note.

Another area which attracts quite a bit of attentions these days is high speed list processing and logic programming for artificial intelligence applications. Here, again, design of special purpose hardware and/or exploitation of parallelism are hot research subjects.

Main purpose of this lecture note is to introduce to the readers research and development activities in Japan in the these two areas. This lecture note consists of four parts. Chapter 2 summarizes the history and current status of Japanese commercial supercomputers in general. Chapter 3 elaborates, in somewhat detail, the technology, architecture and software of the Fujitsu's FACOM Vector Processor System, VP-100 and VP-200, as an example of the state-of-the-art Japanese supercomputers. Chapter 4 states future prospects for supercomputers in the area of high energy physics. Chapter 5 outlines the national projects and some other research activities in Japan. Concluding remarks are given in chapter 6.

References [1,2] are recommended for general knowledge of supercomputers. There is also a lecture note on U.S. supercomputers in this volume.

2. HISTORY AND CURRENT STATUS OF JAPANESE SUPERCOMPUTERS

2.1 History

The first Japanese computer with vector processing capability, Fujitsu's FACOM 230-75 Array Processing Unit (APU) was installed at the National Aerospace Laboratory in 1977. At that time, importance of vector processing was not fully recognized in Japan, and only two systems have been delivered. Around the same period, Hitachi and NEC have also introduced the Integrated Array Processors (IAP's), which can be integrated to their large scale general purpose computers. In this era, a tremendous amount of experiences have been accumulated in vector processing, which lead to the present generation supercomputers in Japan.

From 1982 to 1983, Fujitsu, Hitachi, and NEC announced supercomputers. These are the VP-100 and VP-200 by Fujitsu^{3,4]}, the HITAC S-810/10 and S-810/20 by Hitachi^{5]}, and the NEC SX-1 and SX-2 by NEC^{6]}. The maximum performance of these systems ranges from 267 to 1300 million floating point operations per second (usually abbreviated as MFLOPS).

2.2 Current status

In Japan, large scale numerical computations are required in various areas. At the national level, fusion research, aerodynamics, atmospheric researches including numerical weather prediction, and nuclear engineering are major applications. Satellite picture processing is also expected to be an important application in the near future. In the industry, device analysis and circuit simulations for VLSI design and the finite element method for structural analysis are the typical applications. In academia, basic researches such as quantum chemistry, solid state physics, molecular dynamics and fluid dynamics are typical fields which require large scale number crunching.

As of April 1984, three Japanese vector processors have been installed at national universities in Japan, and are now fully operational: the VP-100's at the Institute of Plasma Physics in Nagoya University and Computing Center of Kyoto University, and a S-810/20 at Computing Center of University of Tokyo. More installations are expected, both domestic and abroad, in the near future. In addition to the above systems, two CRAY 1S and one Cray X-MP systems have been in use in Japan.

Table 1 summarizes the hardware specifications of supercomputers manufactured by Hitachi, Fujitsu, NEC and Cray Research. It is to be noted that all these machines basically employ the pipeline architecture.

The Japanese supercomputers and their manufacturers share some interesting characteristics. Firstly, Fujitsu, Hitachi, and NEC are also

large semiconductor manufacturers in the world, and can utilize the latest semiconductor technology. Secondly, the Japanese supercomputer manufacturers incorporated the vector architecture to their mainframe architecture, since they all have long experiences in developing large scale mainframes. Especially, Fujitsu and Hitachi have adopted IBM compatible instruction set for their general purpose computers. Therefore, good affinity of vector processing capabilities within the framework of general purpose computing environment, such as very powerful and sophisticated vectorizing compiler and ease of use, is a key consideration for these systems.

2.3 Some performance comparisons

Table 2 describes the performances of VP-200, S-810/20, and CRAY X-MP (single CPU) for Livermore Loops as of January 1984. The Livermore loops are collection of kernel loops which have been selected by Lawrence Livermore National Laboratory^{7]}, in order to measure the performance of supercomputers for their typical workloads. In Table 2, The performance of VP-200 was measured by Fujitsu at its Numazu works^{3]}, whereas those for S-810/20 and CRAY X-MP are taken from^{8]}. The NEC SX system is still in the development stage and its performance is not available. Some other performance measurements have been reported in^{9]}.

3. FUJITSU'S VECTOR PROCESSOR SYSTEM: FACOM VP-100/VP-200

In this chapter, the Fujitsu's Vector Processor System, FACOM VP-100 and VP-200, are described, more in detail, as an example of the state-of-the-art Japanese supercomputers.

3.1 Design philosophy of Vector Processor System

Based on the extensive analysis of more than one thousand FORTRAN programs in scientific and engineering applications, the following primary objectives of the FACOM Vector Processor System have been identified during the early phase of development:

- (1) To realize a system with high performance for users' application programs rather than just the peak performance.
- (2) To realize a system which is easy to use for non-specialists in developing application programs.

In the FACOM Vector Processor System, these two objectives have been achieved through the implementation of the following items in both hardware and software.

For increasing effective performance:

- Flexible definition of vectors and efficient vector data editing functions
- Vectorization of the DO loops including IF tests
- Dynamically configurable vector registers
- Concurrent operations at various levels of the system
- High speed scalar unit

For ease of use:

- Various tuning and debugging tools
- Optimization Control Line (OCL) to assist vectorization
- Interactive vectorization capability

3.2 Technology

Fujitsu's latest circuit and packaging technologies have been utilized in the Vector Processor System. Logic LSI's contain 400 gates per chip, with some special functional chips such as register files containing 1,300 gates. Signal propagation delay per gate of these LSI's are 350 picoseconds. Memory LSI's integrating 4K bits per module with an access time of 5.5 nanoseconds are used where extremely high speed is required. Up to 121 LSI's can be mounted on a 14-layered printed circuit board, called Multi-Chip-Carrier (MCC). Logic LSI's and memory LSI's can be mixed on the same MCC. 13 such MCC's are mounted in a $(50 \text{ cm})^3$ cube, called Stack. Forced air cooling technique has been adopted throughout the system. Machine cycle is 7.5 nanoseconds for the Vector Unit and 15 nanoseconds for the Scalar Unit. As for the main storage, 64K bit MOS static RAM LSI's with 55 nanosecond chip access time are used.

3.3 System architecture

Two models, the VP-100 and the VP-200, both employ the architecture with multiple pipeline units which can operate concurrently. As stated in chapter 2, the maximum performances of VP-100 and VP-200 are 267 MFLOPS and 533 MFLOPS, respectively. Figure 1 illustrates the block diagram of VP-200 System. The Vector Processor System consists of a Scalar Unit, a Vector Unit, a main storage unit, and channel units.

3.3.1 Scalar Unit

The Scalar Unit fetches and decodes all the instructions. There are 195 scalar instructions and 83 vector instructions. When an instruction is of scalar type, it is executed in the Scalar Unit, otherwise issued immediately to the Vector Unit. The Scalar Unit is equipped with 64K bytes of high speed buffer storage and scalar registers such as 16 general-purpose registers and

8 floating-point registers. A direct path is provided between the Scalar and the Vector Units, so that the general purpose registers and the floating point registers can be referenced or updated directly from the Vector Unit. The performance of the Scalar Unit is 8.6 MFLOPS for the 14 Livermore kernel loops^{3]}.

3.3.2 Vector Unit

The Vector Unit mainly consists of vector registers, mask registers, two load/store pipelines, an add/logical pipeline, a multiply pipeline, a divide pipeline, and a mask operation pipeline. In the VP-200 System, all the arithmetic pipelines and the mask operation pipeline operate with 7.5 ns machine cycle, and each pipeline can process 2 elements per cycle. The divide pipeline has 1/7 of the throughput of either the add or the multiply pipeline. Any two of these arithmetic pipelines may operate concurrently. All the floating-point arithmetic operations are performed in full precision (64 bits) in the vector unit. Two load/store pipelines are provided to transfer data between Vector Unit and the Main Storage Unit. In order to control conditional vector operations and vector editing operations, bit strings (called mask vectors) are provided. The data in mask vectors consist of "0" and "1", representing logical values (to be denoted as "FALSE" and "TRUE"). The mask operation pipeline performs logical operations associated with the mask vectors.

In the case of the VP-100 System, the throughput of each pipeline is half of that for the VP-200 System.

3.3.3 Main Storage Unit

The Main Storage Unit for the VP-200 System has a maximum capacity of 256M bytes and is interleaved in 256 ways at maximum. Both single precision and full precision floating point words (both in IBM format) can be defined in the Main Storage Unit. Each load/store pipeline has a maximum transfer rate of 32 bytes/15 nano seconds. Again, VP-100 System has half the throughput of that for the VP-200 System.

3.4 Vectorizing compiler:FORTRAN77/VP

Vectorization, in general, refers to the process of generating object codes with vector instructions from the source codes. Since the performance greatly differs between scalar and vector instructions, how much of a code has been vectorized is a very important factor for any vector processor. A vectorizing compiler, FORTRAN77/VP, is available for Vector Processor Systems. FORTRAN77 has been chosen as the language for VP Systems so that the large amount of software assets can readily be available for vector processing. In order to obtain high vectorization ratio for wide range of application programs, FORTRAN77/VP compiler vectorizes not only simple DO loops but nested DO loops and vector macro operations such as innerproduct

and finding the maximum or the minimum value. It can also detect and separate the statements with recurrences. These general techniques have been reported in [10]. The following subsections will focus on the hardware and software aspects which are related to the advanced features.

3.5 Implementation of advanced features

3.5.1 Flexible definition of vectors and vector data editing functions

The Vector Processor System provides the following three modes of memory access:

- Contiguous access (e.g., A(I) in loop over I))
- Constant stride access (e.g., A(2*I), or A(I,J) in loop over J)
- Indirect addressing access (e.g., A(L(I)) where L is an Index List)

The Vector Processor System also provides various vector data editing instructions. Here, only Vector Compress/Expand instructions will be described. Figure 2 illustrates the Vector Compress and Vector Expand operations. The Vector Compress instruction selects the elements of a Vector Register with the corresponding "TRUE" mask bits, then stores them at contiguous locations of another vector with their relative order preserved. The Vector Expand instruction performs the reverse operation of the Vector Compress instruction. The Compress and Expand operations are very frequently utilized in the vectorization of DO loops which contain IF tests, as well as in performing algorithmic shuffling of vector data. In the former case, instructions for these operations are generated by the compiler, whereas in the latter case these operations can be explicitly stated in FORTRAN. For example,

```
J=1
DO 10 I=1,N
IF (M(I)) THEN
  A(J)=B(I)
10 J=J+1
compress array B into array A under logical vector M.
```

3.5.2 Vectorization of DO loops containing IF tests

In order to vectorize DO loops which contain IF tests, FORTRAN77/VP compiler provides the following three alternatives:

- Masked arithmetic operations

Figure 3 illustrates an example of masked Vector Add operation: all the elements in Vector Registers B and C are added, but only the results with the corresponding "TRUE" mask bits are stored in Vector Register A; otherwise old values in A are retained. Note that the total execution time is not affected

by the population of "TRUE" values (the "true ratio") in the mask registers.

- Compress/Expand method

When this method is employed, a mask vector is generated using Compare instruction, and all the associated arrays are compressed prior to the execution of the main body of a loop which is under the influence of the IF test. After these operations the same mask vector is used to expand the results back to original locations.

- Vector indirect addressing method

When this method is employed, a mask vector is generated first and a index list is generated using the mask vector, prior to the execution of the main body of a loop which is under the influence of the IF test.

The FORTRAN 77/VP compiler vectorizes DO loops containing IF tests by using one of the above three methods. The optimal choice may be characterized by two parameters: the true ratio of IF tests, and the access frequency to the Main Storage Unit, which is the ratio of the non-overlapped load/store operation counts to the total operation counts within a loop. The FORTRAN/77 compiler estimates the execution time for three methods and selects the optimal one. When the true ratio, which is not known to the compiler, is crucial in selecting the optimal method, a programmer can provide such information. More about this in 3.6.

3.5.3 Dynamically configurable vector registers

In order to make full use of the vector registers, the Vector Processor Systems provide dynamically configurable vector registers. The basic configuration of vector registers in VP-200 system, for example, is 32 elements x 256 registers, but the vector registers may be concatenated to take such configurations as 64 elements x 128 registers, 128 elements x 64 registers, ..., 1024 elements x 8 registers, at maximum.

FORTRAN77/VP compiler automatically assigns the optimal vector register configurations based on the number of vectors and/or the vector length required to process a loop. When the vector length is too short, load/store instructions would be issued frequently. On the other hand, when the vector length is too long, the number of available vectors would be small while each vector is not fully loaded. A programmer can also provide information on the vector length through the tuning facilities.

3.5.4 Concurrent operations

The Vector Processor Systems allow concurrent operations at various levels. In the Vector Unit, five pipeline units can operate concurrently, as stated in 3.3.2. Within each arithmetic pipeline unit, vector operands associated with consecutive vector instructions can flow without flushing the

pipe completely. The vector Unit and the Scalar Unit can also operate concurrently, as long as the vector and scalar instructions do not have data dependency. Figure 4 shows an example of concurrent operations between the Scalar Unit and the Vector Unit. This figure illustrates how subsequent scalar instructions can be executed before completion of the preceding vector instructions.

The compiler performs extensive dataflow analysis of FORTRAN source programs and schedules the instruction stream so that maximal possible concurrency can be exploited. This process is generally called "parallelization": it includes reordering of instruction sequence, balanced assignment of two load/store pipelines, and serialization of instructions, wherever necessary.

3.5.5. High speed I/O handling

In order to facilitate high speed data transfer between the Main Storage Unit and disk files, parallel I/O function (commonly called disk striping) is provided. Asynchronous I/O is also supported to allow overlapped data transfer and computation.

3.6 Ease of use in program development

The Vector Processor System is designed to be used as a loosely-coupled system with a front end machine. The compilation, linking and editing are performed on the front end machine and only the execution steps are performed on the Vector Processor. In this way, all the file handling capabilities and other general purpose features of the front-end machine can be fully utilized. Coupling of the two machines can be done via shared files, channel-to-channel adaptor(CTC) or both.

In order to facilitate the program development and tuning, the various software tools are provided. Some of them are described here. It is to be noted that tuning refers to modification of programs within the context of FORTRAN77 and no special function call is involved.

- Optimization Control Lines (OCL)

There are cases where the compiler can generate object codes with higher performance if the programmer supplies information to the compiler. Special commands called Optimization Control Lines may be inserted in FORTRAN source codes in the form of comment lines. Some examples are: specifying the true ratio of an IF test, specifying the vector length of a DO loop for better utilization of vector registers, and forcing vectorization of a loop when it apparently has a recurrence due to unknown parameter values but not really the case.

- Interactive Vectorizer

Interactive Vectorizer is provided for tuning programs from a display terminal. A programmer can know whether DO loops in the program are vectorized, how to improve performance and other information. A programmer can also insert Optimization Control Lines, modify the source program, and enter commands on the same CRT screen interactively.

- Scientific Subroutine Library (SSL-II/VP)

Scientific Subroutine Library (SSL-II/VP) provides wide range of basic subroutines which can be used as building blocks for application program development. These subroutines are all coded in FORTRAN and highly vectorizable. Some examples are: matrix calculations, simultaneous linear equations, algebraic eigenvalue problems, Fourier and Laplace transforms, ordinary differential equations, special functions and random number generation. Some of these subroutines employ vector algorithms which are highly optimized for the Vector Processor architecture, as described in 3.8.

3.7 Applications

As stated earlier, the Vector Processor Systems can be used for wide range of applications. In Table 3, execution time in both scalar and vector modes are shown for some applications^{3]}. The first three are taken either from subroutine packages or part of programs, while three others are complete programs.

3.8 Importance of vector algorithm development

As described in ^{1,2]} there exist several basic numerical algorithms which yield very high performance on vector machines. It should be noted that the optimum vector algorithms still depend on particular architectures, such as the available main storage and vector register sizes, definition of vectors, and type of available arithmetic concurrency and vector data editing functions. Therefore, it is important to develop basic numerical algorithms which are tailored for the VP-100/VP-200 Systems. Some such examples are summarized in the following ^{11]}. All the codings have been done in FORTRAN.

- Triangularization of Symmetric Matrices

The Cholesky Decomposition scheme using middle product rather than inner product has been found to be the optimal algorithm. For example, 430 MFLOPS have been achieved for decomposing a 1024 by 1024 matrix.

- Fast Fourier Transform

Several radix-2 Fast Fourier Transform schemes have been examined for full precision complex data, and Stockham's self-sorting variant using the vector indirect addressing function has been found to be the optimal one. For example, a 4096 point transform takes 973 microseconds, which corresponds to 227 MFLOPS (Also in Table 3).

- Random number generation

A recurrence-free mixed congruential method has been implemented to generate uniform pseudo-random numbers. The rate is about 83 million random numbers per second, when 16384 floating point random numbers are generated.

All the above algorithms have been incorporated in SSL II/VP.

4. APPLICABILITY OF SUPERCOMPUTERS TO HIGH ENERGY PHYSICS

As for the applicability of supercomputers for high energy physics in general, three major areas and their prospects may be stated as follows.

4.1 Monte Carlo simulation for lattice gauge theory

Several published papers^[12] as well as investigation at Fujitsu indicate that supercomputers are very powerful tools for the lattice gauge simulation. Although the original scalar code consists of heavily nested loops with recurrences, code can be restructured to yield loops with very long vectors (say more than several hundreds). It has also been recognized that vectorized random number generator is important in this type of applications.

4.2 Monte Carlo event simulation

The Monte Carlo Method in this category is very much different from the above example. Each particle to be simulated behaves completely independently of others (except that reactions may produce new independent particles) but in quite random fashion. Similar problems are encountered in neutron transport calculations in the nuclear engineering. This type of problem, which might be dubbed "Event-oriented parallelism", is hard to vectorize automatically, since the code is scalar in nature and heavily populated with IF tests.

Conventional hand-vectorization technique is to process many particles in one pass, and to form vectors with particles possessing identical characteristics, by utilizing vector editing functions such as vector indirect addressing and Compress/Expand operations as stated in 3.5.1. Capability of the compiler to vectorize IF tests is also recognized as very important.

There are several difficulties in hand-vectorizing this type of codes. Firstly, code-restructuring involves deep understanding of the code itself and takes considerable amount of efforts. (Monte Carlo codes are usually very large in size). This also raises an issue on the consistency with existing codes, which may be still evolving. Secondly, the efficiency of vector processing really depends on the varieties of reaction patterns and/or

complexities of geometry, which strongly influence the effective vector length. It should also be pointed out that transportability of a vectorized code to smaller machines is not trivial even though it is written in FORTRAN, since vectorized code usually takes considerably larger memory space than the original scalar version.

In spite of the above drawbacks, however, the author is conducting researches at Fujitsu to establish methodologies to vectorize "Event-oriented" Monte Carlo codes in general, since this is a very important application field. So far, he has taken FOWL code from CERN to experiment with, and obtained more than three times performance improvement over scalar versions. Further efforts are being continued.

4.3 Off-line data analysis and track reconstruction

It is generally recognized that almost 90 percent of the computation time in high energy physics is consumed in this area. Experiences at Fujitsu, as well as others', to automatically vectorize the original scalar codes have not been fruitful. The very nature of the computation is again "Event-oriented parallelism", since the behavior of each event is completely independent of others but highly irregular. Even if restructuring approach is taken, which will be similar to the one for Monte Carlo event simulation, the resulting performance improvement is not obvious, and hand-vectorization effort (code size is enormous) and transportability and/or consistency with existing codes will remain big obstacles again. An early attempt to process many events on Cray-1 has been reported in [13]. Development of entirely new vector algorithms to process one event may be a possible solution, but nobody has made any breakthrough yet to date. It is of the author's opinion, however, that further research should be conducted in this area.

5. NATIONAL PROJECTS AND OTHER ACTIVITIES IN JAPAN

This chapter outlines national projects and some other research efforts in Japan in the area of number-crunching and logic and symbolic processing. Although there are numerous research projects going on in Japan, only a few of them are described here due to space limitation.

5.1 National Super-speed Computer Project^{14]}

Under the auspices of the Ministry of International Trade and Industry (MITI), a national project started in 1982 to develop the High-Speed Computing System for Scientific and Technological Use (also called Super-speed Computer System).

In order to meet the anticipated requirement in 1990's in the application areas of national interest, the target of this project has been set to develop a computer system with 10 GFLOPS performance by the end of FY1989 (i.e., March 1990). This project involves research and development in such areas as parallel architecture and new devices.

In the area of parallel architecture, research and development activities include the study of various parallel architectures, hierarchical memory structure to meet the demands of high data transfer rate as well as the software development to efficiently utilize such high performance system. A dataflow machine, called SIGMA-1, is also under development at Electrotechnical Laboratory^{15]} as a part of this project.

In the area of the new devices, research and development activities are conducted in Josephson Junction Device, High Electron Mobility Transistor (HEMT) device, and Gallium Arsenide (GaAs) MESFET device. HEMT is a variation of GaAs device which has a heterojunction structure (superlattice) in the active area, so that the electron mobility is higher than that for the conventional GaAs device, especially at liquid nitrogen temperature. This device was developed in Japan in 1980. Most recent development of HEMT devices includes the frequency divider which operates at 8.9 GHz (equivalently, 22 picoseconds/gate with 2.8 mW power dissipation per gate), and 4K bit Static RAM with 2.0 nanosecond access time and 1.6 W power dissipation per chip. In both measurements, devices are chilled at liquid Nitrogen temperature. Other devices are also making progress.

The main applications for the final system will include plasma simulations for fusion research, atmospheric research, aerodynamic computations (Navier-Stokes Equation solver) and satellite picture processing. It is to be noted that the final system is regarded as a scientific demonstration system and further refinement will be necessary to make it a commercial product.

5.2 Microcomputer Array PAX^{16]}

PAX-128 (Processor Array eXperiment) has been developed at Tsukuba University by Prof. T. Hoshino and his research group. It is a evolutionary model of PACS-9 (9 processors), PACS-32 (32 processors) and PAX-32 (also 32 processors) which were previously developed. The PAX-128 mainly consists of one Control Unit and 128 Processing Units. Each Processing Unit is built with Motorola 68B00 microprocessor (2 MHz clock), AMD Am9511A-4 microprocessor (4 MHz clock) and 28 K Bytes of local memory. These processing Units are connected with two-dimensional Nearest Neighbor Mesh Network (8 x 16).

Several application programs have been implemented on this system: partial differential equation model in aerodynamics, Poisson equation solver, nuclear reactor computation, Monte Carlo simulations of plasma particles and two dimensional spin model. The above models all exploit the proximity interactions which are inherent in the physical processes. Also some programs for non-proximity type models have been implemented, such as Monte Carlo particle simulation with general interactions, linear equation solver and FFT. It is claimed that all these programs exhibit high degree of parallelism, and consequently high utilization of processors. It is reported that about 4 MFLOPS performance has been achieved for certain application.

5.3 Fifth Generation Computer Systems^{17,18,19]}

In April 1982, Institute for New Generation Computer Technology (ICOT) began research and development for Fifth Generation Computer Systems (FGCS) under the auspices of Ministry of International Trade and Industry (MITI). The fifth Generation computers are defined as the computers which will be used predominantly in 1990's. In future, non-numeric data processing, including symbol processing and applied artificial intelligence, are expected to play more important roles than at present. Possible application areas will be: various kinds of expert systems, natural language processing, speech processing and graph/image processing, just to name a few.

The functions of the Fifth Generation Computer Systems may be roughly classified as: problem solving and inference, knowledge-base management, and intelligent interface. In order to realize such functions, the knowledge information processing systems will be developed based around inference machines, knowledge-base machines and intelligent man-machine interface machines.

This project will span a period of ten years: three years for the initial stage, four years for the intermediate stage, and three years for the final stage. The research goals for the initial stage are:

- to investigate architectures suitable for logical inference and database operations,
- to design a Prolog-based kernel language for knowledge representation and inferencing,
- to design workstations containing a sequential inference mechanism.

During the initial stage, the foundations for the entire project will be established, and the basic tools to be used in the intermediate stage will be built. The ideas produced in the initial stage will be prototyped in the intermediate stage, and the researches in the intermediate stage will be integrated and evaluated in the final stage.

Inference machines and logic programming

One primary focus on the FGCS project is to develop a hardware mechanism which supports the basic functions for inferencing. In this project, LIPS (Logical Inferences Per Second) is used as performance measure for inferencing. It corresponds to 100-1000 instruction per second, if implemented on the conventional machines. It is estimated that the performance requirement in 1990's will be as high as 100 MLIPS to 1 GLIPS. A prototype sequential inference machine developed in the initial stage (PSI) is rated as 30 KLIPS. An extension machine is being developed, which is expected to give 150 KLIPS. In the next stage, Parallel Inference Machine (PIM) is to be developed. Several types of parallel architecture have been proposed for PIM.

The inference machine architecture should be capable of processing knowledge information. Most of the artificial intelligence software has been based on LISP. In this project, it is claimed that Prolog-like language is more powerful and suitable for parallel processing of knowledge information than LISP-like language. In the initial stage, a sequential inference language, KL-0, has been developed for the PSI machine. The specifications for KL-1, which incorporates parallelism, will have been completed by the end of FY1984; it is to be run on PIM machine in the next stage. KL-2 is to follow KL-1 as the final language in this project.

Knowledge-base systems

A knowledge base is an extension of a database, that is, a database plus a collection of rules with which to manipulate the database for inferencing. In this project a parallel relational database machine is regarded as an appropriate starting point for knowledge-base machine. In the initial stage, a prototype relational database machine, called Delta, has been developed. Within Delta, up to four relational database engines can run in parallel. Currently, several PSI's have been hooked up to Delta via a local area network.

This project is to complete its initial stage in March 1985. In November 1984, the International Conference on Fifth Generation Computer Systems was held in Tokyo with more than 1000 attendees, and both PSI and Delta were successfully demonstrated at the conference site and at ICOT, respectively. There are also national projects on artificial intelligence currently going on in the U.S.A. and in Europe. Research activities in this area really seem to be expanding worldwide.

5.4 FLATS: A Machine for symbolic and algebraic manipulation^{20]}

Recently, there has also been a great demand in scientific and engineering communities for computer systems which can perform high-grade formula manipulations, such as processing of complicated algebraic

expressions, differentiation and integration operations at high speed. All the existing formula and symbol manipulation software have been implemented in LISP, and are not fast enough on general purpose computer systems when it comes to practical applications.

FLATS system was developed at Institute of Physical and Chemical Research, under the direction of Dr. E. Goto, in order to fill the above mentioned gap. The project started in 1979, and completed in 1984. FLATS stands for Formula, Lisp, Associative computing, Tuple and Set, and refers to a computer system for exclusive use in formula and symbol manipulation. A formula manipulation system REDUCE^{21]} is to be transplanted on the FLATS system.

The target of this project was to develop a system which outperforms large scale mainframes for list processing applications. ECL 10K and 100K logic families have been adopted for logic elements. FLATS hardware has the following several novel features:

- (1) Parallel Run-time Data Type Checking,
- (2) High Speed Execution of LISP Instructions such as CAR, CDR and CONS,
- (3) Associative Processing with Parallel Hardware Hashing,
- (4) Separate Cash memories for Instructions and Data,
- (5) Hardware Garbage Collection Instructions.

This project was jointly conducted by the Institute for Physical and Chemical Research, Mitsui Engineering and Shipbuilding Company, and Fujitsu Limited. The actual construction of the system was done by Mitsui. RAND Corporation (U.S.A.) and Cambridge University (U.K.) also participate in this project in the area of software and algorithm development for formula manipulation. At present, performance evaluation of FLATS machine is being continued at the Institute for Physical and Chemical Research.

6. CONCLUSIONS

This lecture note has summarized the development history and current status of Japanese supercomputers, and also described the Fujitsu's Vector Processor Systems as an example of the state-of-the-art technologies of Japanese commercial supercomputers. This lecture note also has outlined some national efforts to develop high speed computers for both number-crunching and logic and symbolic processing. Demands for large-scale computations are increasing in Japan in various scientific and engineering fields. The author sincerely hopes that this lecture note serves the purpose of introducing Japanese activities to the readers.

ACKNOWLEDGEMENT

The author wishes to thank Mr. K. Uchida, Mr. Tanakura, Dr. Tago of Fujitsu Limited for providing materials on the Vector Processor System. The author also wishes to thank Mr. D. Lord and Mr. F. James of DD Division, CERN for providing to the author FOWL code and other high energy physics codes.

* * *

REFERENCES

- [1] Hockney, R. and Jesshop, C.: Parallel Computers, Adams-Hilger, 1982.
- [2] Zacharov, V.: Parallelism and Array Processing, Lecture Note at 1982 CERN School of Computing, Zinal, Switzerland, also IEEE Trans. Computers, Vol. C-33, No. 1, January, 1984, pp. 45-78.
- [3] Miura, K. and Uchida, K.: The FACOM Vector Processor VP-100/VP-200, Supercomputers: Design and Applications (Edited by Hwang, K.), IEEE Computer Society Press, 1984, pp. 127-138.
- [4] Uchida, K. and Itoh, M.: High Speed Vector Processors in Japan, Submitted to Conference on Vector and Parallel Processors in Computational Science II, held in Oxford, U.K., on August 28-31, 1984. Proceedings to appear in Computer Physics Communications.
- [5] Nagashima, S. and Inagami, Y.: Design Consideration for A High Speed Vector Processor: The Hitachi S-810, Proc. International Conference on Computer Design, IEEE Computer Press, 1984, pp. 238-243.
- [6] Watanabe, T.: Architecture of Supercomputer --- NEC Supercomputer SX System, NEC Research and Development, No. 73, 1984.
- [7] Riganati, J. and Schneck, P.: Supercomputing, IEEE Computer, Vol. 17, No. 10, October, 1984, p. 112.
- [8] Fuss, D.: A Comparison of Japanese and American Supercomputers, The NMFECC Buffer, January, 1984.
- [9] Mendez, R.: Benchmarks on Japanese and American Supercomputers -- Preliminary Results, IEEE Trans. on Computers, Vol. C-33, No. 4, April 1984, p. 374.
- [10] Kamiya, S., Isobe, F. and Takiuchi, M.: Practical Vectorization Techniques for the 'FACOM VP', Information Processing, (Edited by Mason, R.F.A), IFIP, 1983, pp. 389-394.
- [11] Matsuura, T. Miura, K. and Makino, M.: Supervector Performance without Toil ---FORTRAN Implemented Vector Algorithms on VP-100/VP-200, Submitted to Conference on Vector and Parallel Processors in Computational Science II, held in Oxford, U.K., on August 28-31, 1984. Proceedings to appear in Computer Physics Communications.
- [12] Barkai, D. et al.:A Highly Optimized Vectorized Code for Monte Carlo Simulations of SU(3) Lattice Gauge Theories, Proc. International Conference on Parallel Processing, IEEE Computer Press, August 1984, pp. 101-108.

- [13] Zacharov, V.:Invariant Coding and Parallelism in Data Processing, CERN Report DD/82/1, January, 1982.
- [14] Kashiwagi, H.:Japanese Super-speed Computer Project, High-speed Computation (Edited by Kowalik, J.), NATO ASI Series Vol. F7, Springer-Verlag, 1983, pp. 117-126.
- [15] Hiraki, K. et al.:A Hardware Design of the SIGMA-1, A Data Flow Computer for Scientific computations, Proc. International Conference on Parallel Processing, 1984, IEEE Computer Society Press, pp. 524-531.
- [16] Hoshino, T. et al.:Highly Parallel Processor Array "PAX" for Wide Scientific Applications, Proc. International Conference on Parallel Processing, IEEE Computer Press, August 1983, pp. 95-105.
- [17] Motooka, T.: Japanese Project on Fifth Generation Computer Systems, High-speed Computations (Edited by Kowalik, J.), NATO ASI Series, Vol. F7, Springer-Verlag, 1983, pp. 99-116.
- [18] Motooka, T. and Stone, H.:Fifth-Generation Computer Systems:A Japanese Project, IEEE Computer, Vol.17, No.3, March,1984, pp.6-13.
- [19] Proc. the International Conference on the Fifth Generation Computer Systems 1984, Institute for New Generation Computer Technology, November 6-9,1984, Tokyo, Japan, published by OHMSHA, 1984.
- [20] Goto, E. et al.:Design of a LISP Machine -- FLATS, Conference Record of 1982 ACM Symposium on LISP and Functional Programming, 1982.
- [21] Hearn, A.:REDUCE-2 User's Manual, UCP-19, Univ. of Utah, 1973.

Table 1 Comparisons of supercomputers

Model	FUJITSU VP- 100 200		HITACHI S810/ 10 20		NEC		CRI CRAY X-MP (2 CPU's)
Announce- ment	3Q/1982		3Q/1982		2Q/1983		2Q/1983
First Customer Service	4Q/1983		4Q/1983		(1985)		4Q/1983
Machine Cycle (nsec)	7.5		14		7	6	9.5
Peak Per- formance (MFLOPS)	267	533	315	630	570	1300	420
Vector Registers (KB)	32	64	32	64	40	80	8
Mask Registers (b x No.)	16 x 256	32 x 256	256 x 8		128 x 8	256 x 8	64 x 8
Main Storage Size(MB)	128	256	128	256	256		32
Main Stor- age RAM (Kb/chip)	64 MOS Static		16 MOS Static		64 MOS Static		4 Bipolar
Max. I/O Transfer Rate(MB/s)	96		96		50		224
No. of Channels	16/32		8/16 24/32		32		16
Cooling	air		water (vector unit)		water		freon

Table 2 Performance measurements for Livermore Loops

Loop Number	FUJITSU VP-200	HITACHI S810/20	CRI CRAY X-MP (1CPU) (CFT1984)
1	331.4	228.0	153.0
2	180.4	239.4	76.6
3	338.2	211.9	95.8
4	88.1	59.2	41.1
5	10.0	5.4	8.7
6	9.5	4.6	8.0
7	331.0	232.7	167.9
8	90.4	48.8	95.7
9	260.8	207.6	163.0
10	85.9	49.0	59.9
11	4.8	9.8	3.1
12	115.3	83.0	76.5
13	6.2	4.2	4.8
14	13.8	8.5	6.9
Arith. Ave.	133.3	100.2	68.6

(Units: MFLOPS)

Table 3 Performance Measurements for Some Application Programs on VP-200

Program Number	Description of Programs	Computation Time		Performance Ratio
		Scalar (Sec.)	Vector (Sec.)	
1	Matrix Multiplication (Order: 100)	307.66	4.08	75.4
2	Linear Equation Solver (Order: 100)	.141	.01	14.1
	(Order: 256)	2.27	.056	40.6
3	Self-sorting Type FFT (4096 points, Radix 2)	---	.973E-3	(227 MFLOPS)
4	Molecular Dynamics (Molten Salt)	144.22	9.97	14.5
5	Simplified Marker and Cell Method	137.0	15.8	8.7
6	Image Processing for Synthetic Aperture Radar(100Km x 100Km)	1.44E4	540	27

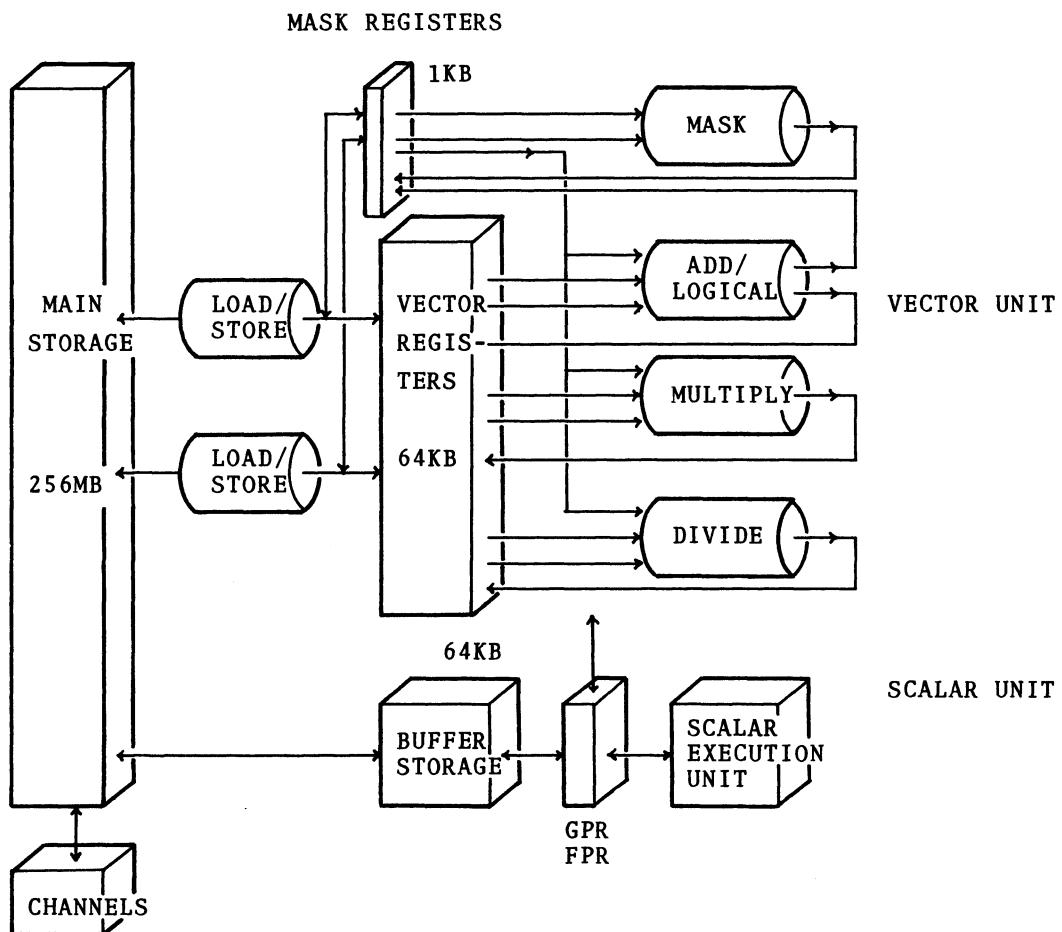


Figure 1 Block diagram of FUJITSU Vector Processor System
VP-200

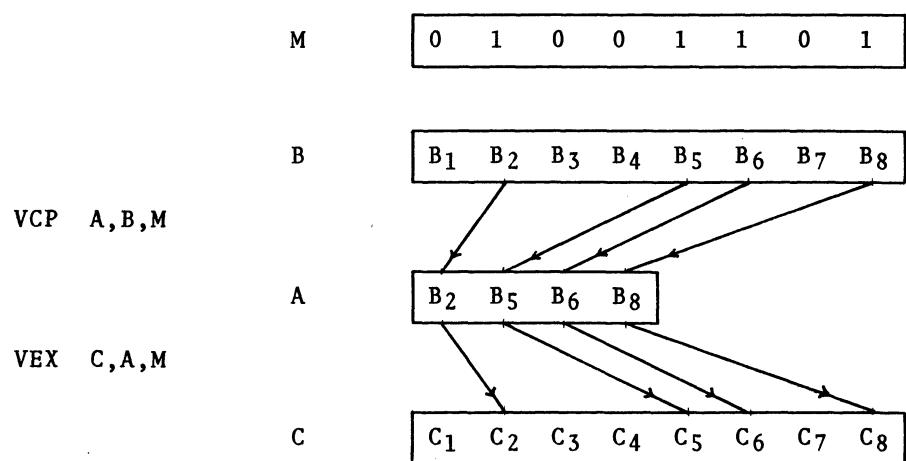


Figure 2 Vector compress and vector expand operations

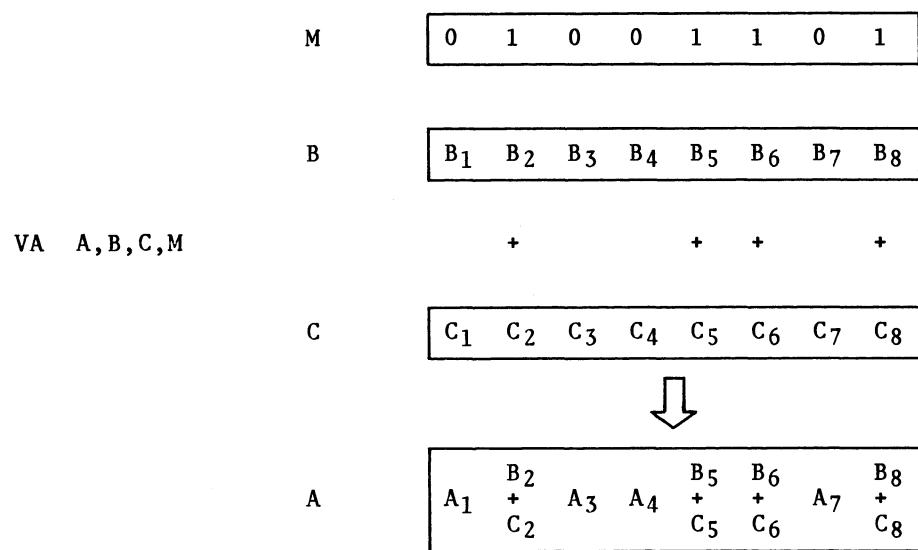


Figure 3 Masked Arithmetic Operation
(Vector C is added to Vector B under mask M.)

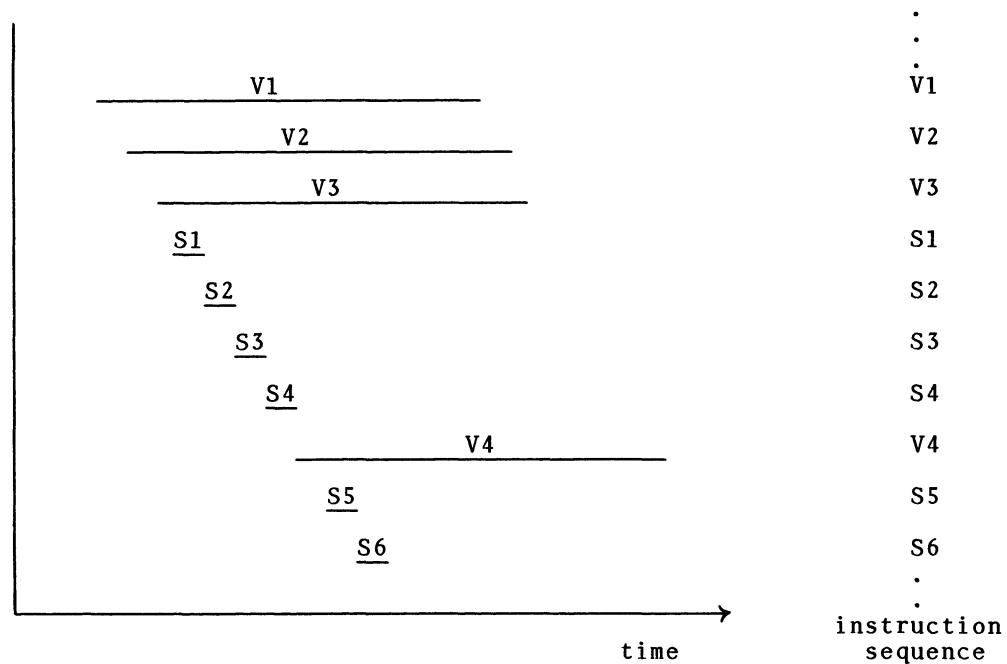


Figure 4 Concurrent Operations of Vector and Scalar Instructions

APPLICATION OF VLSI IN HIGH ENERGY PHYSICS

B.D. Hyams
CERN, Geneva, Switzerland

INTRODUCTION

My talks will be concerned with the introduction of specialized chips into high energy physics. In fact my topic is really specially designed chips, not VLSI chips.

I'm probably the only lecturer at this school who is not an expert in the field about which he lectures.

I will therefore start off my talk by mentioning some reasons why it is only now that the high energy physics community is getting involved in this subject.

The two lectures will then be concerned with the following topics:-

The need for custom chips in this field.

Some areas where it appears possible to apply them fruitfully.

The range of techniques available, and possibilities and limitations for undertaking this kind of work in Europe.

A specific project on which I have worked.

The reason, then, why there is not an expert in this field here is that it is only just starting, and it appears to have been difficult to find an expert in the high energy physics community. In the course of the last year this has probably ceased to be true. VLSI has been around for a number of years, though the definition of what is Very Large Scale Integration continues to be revised almost annually. It has reached high energy physics (in the sense of producing specialized chips) only recently, for a variety of reasons.

One is that there is always a delay in knowledge spreading around. But there are others too. It was only a few years ago, in 1979, that students (in the USA) could design chips, and have them fabricated as part of their course work. This was introduced by Mead and Conway along with a general attempt to introduce simple systematic procedures for chip design. Then such design could move out of the hands of engineers in a few large firms to a much wider group of designers. However this still required factories to fabricate the designs. In the last five years or so a number of U.S. firms (nearly all in Silicon Valley) have started to manufacture chips at a cost economic even for small production runs. I will return to this topic.

THE NEED FOR VSLI

The applications for VSLI are potentially anywhere we are now using electronics. At the output of our detectors, in data compression and transmission, in data selection and processing.

Since about one half of the participants of this school are working in high energy physics, my topic is relevant to the school, even though it is one step removed from computation.

The other half of this group is not involved in high energy physics. So as to give some idea of our problems, I will describe briefly an existing large high energy physics experimental apparatus: UA1, the largest experiment at CERN (Fig. 1), one of the two experiments which have recently shown the existence of the Z^0 and W^\pm bosons.

The main features of this apparatus are:-

- 1) A central set of drift chambers using 6000 wires, for measuring the tracks of charged particles.
- 2) Lead scintillator sandwiches measuring electron and γ -ray energies.
- 3) Iron scintillator sandwiches measuring hadron energies.
- 4) Drift tubes identifying muon tracks.

An event demonstrates the complexity of the information to be recorded (Fig. 2).

This apparatus collects information from some 210^4 channels of electronics. A typical event contains ~ 100 kbytes of information after hardware compression from some 1.7 Mbytes. Some 5 events can be recorded per second. These 5 events must be selected for some 10^4 sec^{-1} occurring in the apparatus. On an IBM 3081 it takes some 10 seconds CPU time to process an event.

A number of detectors of comparable size are now being designed or constructed. There are at least three in the USA, at least four in Europe.

Let us consider the overall economics of a LEP detector now under construction. For example, DELPHI will spend 52% of its budget on mechanics and 48% on electronics -- not including mainframe computing facilities. The apparatus will include some 120,000 channels of electronics. It is clear from these figures that improvements in electronic design will have a major impact on such a project.

What determines the number of electronic channels wanted?

Typically, in a hadron jet of total energy ~ 40 GeV, the "debris" (the emerging hadrons, γ 's from π^0 decays etc.) diverge with angles of 10 to 100 mrad. So that a granularity of better than 10^5 elements is required over 4π radian.

For discrete cells of γ and hadron calorimeters it would be useful to have 210^5 cells. While an array of 300×300 strips do define $\sim 10^5$ cells, the ambiguities in reconstruction have in practice always rendered such schemes very ineffective. Drift chambers and Time Projection Chambers are used to reduce the number of discrete channels, this exchanges many discrete channels for a smaller number, each transforming spatial displacements into displacements in time. However, these devices are inherently slow (because of the stretching of information into time delays) and could not be used around high luminosity machines. In any case TPC's now being constructed are required to measure co-ordinates over a few meters with a precision of less than a millimeter for momentum determination. These end up requiring $\gtrsim 20,000$ electronic channels for one such instrument.

Thus the physics of the experiments would profit by even more electronics than is now envisaged, and the large number of channels suggest the use of specialized circuits to reduce both costs and the mass of the electronics.

POSSIBLE APPLICATIONS OF LSI

In general custom LSI circuits may be economic to replace discrete components where the quantities exceed $10^4 - 10^5$ circuits.

Apart from reducing costs, compact specialized circuits at the detector may (by multiplexing) reduce the number of cables leaving a detector. While a twisted pair of cables occupies a few square millimeters some 510^4 pairs are not only expensive but also bulky and massive. They may seriously reduce the efficiency of the detector by their presence.

An extreme case of this situation arises in the attempt to use silicon strip detectors near a beam pipe. These detectors have $\sim 5 \mu\text{m}$ spatial resolution, but achieve this by using arrays of diodes on silicon with $\approx 20 \mu\text{m}$ pitch. Without integrated circuit electronics the cable volume is ~ 100 times greater than the detector, the volume of "conventional" electronics is some 10^4 times greater than the detector.

As we move along the path of data flow we encounter more areas where improvements would be welcome.

For example -- the recording of signals for a Time Projection Chamber now require a costly chain of electronics. A pulse train, some $20 \mu\text{sec}$ long must be recorded in $\sim 50 \text{nsec}$ bins. There are a number of possibilities for improvements on existing schemes. Moreover some small devices require this pulse train to be scaled down to $2 \mu\text{sec}$ recorded in 5nsec bins, for which no commercial circuits are now available.

Having acquired this vast amount of information we must make a rapid real time selection of "promising" events since less than one in 10^3 can now be recorded. At present the physics interest in $p\bar{p}$ collisions centres around events with large energy flow transverse to the colliding beams.

However one very desirable signature would be events with particles moving ~ 1 mm away from the $p\bar{p}$ collision point, and then decaying. At present such events can only be recognized after all tracks are reconstructed. So there is much room for special purpose electronics for the rapid calculation of recognition algorithms.

Finally it is clear that if an experiment records 10 events per second, and requires 10 seconds to process an event it will require either the full use of 100 IBM 3081's -- or some new methods of processing data. New computing methods will certainly require new electronics.

THE RANGE OF TECHNIQUES AVAILABLE

PAL's

While VSLI is the catchword let me first mention that for some purposes very useful integrated circuits can be made with modest means.

The simplest technique available is that of Programmable Array Logic. These are simply arrays of AND and OR gates, with a few registers, input and output buffers.

The basic logic element is a set of and-or circuits with fusible links to provide AND, OR, XOR circuits (Figs. 3,4)..

The circuits are programmed by burning fusible links. Typically they will replace 5-10 conventional chips (gates, latches). This is a modest gain. But already at this level software for design is desirable (and available), to minimize logic equations, verify functions. Programs are available for up to ~ 10 devices. The chips cost $\sim \$20$ a piece and their fuses can be burnt in on a simple device as for a ROM.

Linear "Semi-Custom"

The simplest solution beyond discrete components for linear circuits is a meccano-like chip with some 150 bipolar transistors and 300 resistors. All these are on one of half a dozen standard chips. The designer has to assemble these components (in practice up to $\sim 70\%$ of them might be used) with one layer of interconnections which the manufacturer provides with a single mask (Fig. 5).

This may seem a rather trivial device at first sight, but it can achieve performance nearly impossible with discrete components. I mention one example -- a fast logarithmic (DC) amplifier accurate to 1% over six decades, working over a wide temperature range.

Typically such a semi-custom chip would cost $\sim \$5,000$ to develop and cost \$3 per piece. In quantities above 50,000 it would pay to make a "custom" designed chip -- i.e. an optimized layout of just the components you need. But for our scale of work "semi-custom" chips will be more economic.

Digital Semi-Custom

Some techniques allow 2 layers of interconnections, with up to 520×5 output gates (Integrated Injection Logic). Typically these may replace 10-20 standard chips.

Chips with bipolar (linear) transistors and a few hundred logic gates are also made.

CMOS "semi-custom" chips with ~ 20 logic cells and ~ 30 flip flops are available. These can operate up to ~ 10 MHz.

Gate arrays with about 6000 gates can be available in $3 \mu\text{m}$ line width chips. The use of silicon is clearly not optimal. But the short turn-around time is a major benefit. E.g. the Sinclair Timex 2x81 P.C. uses 4 gate array chips -- for a production of over 10^6 machines.

Standard Cells

The next level of complexity is to "assemble" a chip design from a set of standard cells. Figure 6a shows an example of one such half cell, in ECL technology. One layer of interconnects can turn this into a variety of circuits equivalent to a standard "chip". Figure 6b shows one such assembly. The manufacturer has a library of some 100 possible circuits per cell.

A chip contains more than 50 such cells, and input and output buffering to the pins. The interconnections between cells are made with a second layer of metallization.

The designer has then to choose an optimum realization for his logic, simulate the logical and timing performance, knowing the individual cell average and worst case performance. He has also to provide a test procedure for the design, and for the production chips. The distribution of heating has to be calculated.

Programs for all this design work are provided by the manufacturer, including routing programs for the cell interconnections.

"Custom" Chips

Finally you may design any device you wish -- taking a silicon wafer, a semiconductor physics treatise and a computer. However to get it fabricated you will still be limited by the technologies available to you.

Here is a list of some of the firms (Silicon foundries in their jargon) in the U.S. who advertise that they will fabricate small quantities of chips to your design -- but clearly with their process. So in fact your starting point will be determined by the firm you choose, or non-industrial lab to which you have access. (Note -- such a list appears about once a year in the journal "VLSI".)

SOME RECENT PROJECTS

CERN Standard Cell Design for UA1

This chip was designed to improve the performance of the UA1 wire chamber recording system. It enabled an existing running system to buffer extra data without rebuilding the electronics. It replaced 25 standard IC's (10K and 100K families) with one quarter of their power consumption. The saving in power is often helpful not for reducing the power bill but for preventing the electronics glowing red hot.

This work was started as a first project, it took 5 weeks of design time -- mostly in designing adequate test inputs. A series of 700 chips were delivered in 7 months. They are now installed and working.

Stanford Time Sampler

An application I have referred to has been the design of a device to digitize 5 nsec slices of a wave form. Such a chip appears to be used in a commercially available oscilloscope, enabling single pulses to be recorded with this resolution. The chip is not available commercially. The Applied Electronics Lab. at Stanford have designed an NMOS chip to do this. The principle is simply to open in sequence a set of 256 switches each for 5 nsec to charge up sequentially 256 storage capacitors (Fig. 7). The voltage levels are then digitized at leisure -- that is at ~ 1 MHz. The chip exists, and is under test. It is hoped to achieve 12 bit precision for the wave form analysis. It will be used at the SLAC linear collider. This is a state of the art device -- fully "custom" designed, by an expert.

A Preamplifier for a TPC

Members of the ALEPH collaboration (a LEP detector) are designing, with the Dortmund University Institute, a monolithic preamplifier for their TPC. Their incentives are small volume, high reliability, low power, low price. The chip should be made with MOSFET technology. It is under study -- time scale one or two years.

Multiproject - 4 Different Chips

A collaboration of Collège de France, LAPP, LPNHE-PARIS VI and LEPHE-X have combined to have 4 different devices developed cooperatively. The devices are to be built in ECL semi-custom gate array technology, each one's development will cost ~ 20 KSF. The devices have the following functions:

Fast track recognition by treating data from a 3x6 array of detectors, and "searching" for one or two tracks.

A comparator with programmable window to search for signals correlated in time (and therefore space) from a TPC.

A generator of protocol signals for Fastbus.

A control circuit for a generator of programmable fast wave forms. This should duplicate real detector wave forms.

Work on this project was started in April '84. Chips are expected in September '84.

Silicon Strip Readout

Two projects are under way to attempt to match low noise preamplifiers to the $\sim 20 \mu\text{m}$ pitch of silicon strip detectors.

One has been initiated by physicists at the MPI, the design and fabrication being carried out at Dortmund.

One has been initiated by a CERN, Hawaii University collaboration, the design and fabrication being done at Stanford.

The Stanford project has produced a first batch of working chips which are now under test. I will describe this project in more detail.

Let me conclude this survey of projects in progress by insisting that it is very incomplete. I am not informed about some projects I know to exist. I am pretty sure that some work is under way with the NORCHIP Scandinavian project. There are probably a number of which I am not even aware. However high energy experimentalists are now moving into this field, both in Europe and the USA.

Silicon Strip Detector Readout

The motivation for this project is from the desire to increase the measuring accuracy of detectors from $50\text{--}100 \mu\text{m}$ (wire chambers) to $3\text{--}5 \mu\text{m}$ for silicon strip detectors.

A silicon strip detector is a piece of high resistivity silicon $\sim 300 \mu\text{m}$ thick and some 5 cm long $\times 2 \text{ cm}$ wide with a diode structure covering it. The diodes can be as narrow as $20 \mu\text{m}$ wide, and they collect the charge from particles traversing the structure. A signal consists of collecting some $2.5 \cdot 10^4$ electrons, giving a few millivolts voltage swing.

The problem is that while these detectors work well the cost per channel has been comparable with other detectors, around SF 50-100 per channel (i.e. per $25 \mu\text{m}$ of width covered). The bulk of the cables to conventional electronics, to say nothing of the electronics itself, has excluded the use of these devices to cover a large area.

So, independently, two groups have started to use LSI technology to solve this problem. I'll describe the project on which I have worked.

While in principle we would like to put silicon amplifier circuits on the same chip as a silicon strip diode particle detector, there are many problems in so doing. The silicon resistivities differ by more than a factor of 100, the two technologies use wafers cut along different crystal axes, the processing

temperatures are very different etc. etc. Finally, there is the basic question of yield. If a device $3 \times 3 \text{ mm}^2$ has a probability of 30% of being defect-free, a $10 \times 10 \text{ mm}^2$ device might be expected to have a yield of 10^{-5} !

So we were driven to the conclusion that we'd have an easier task in designing electronics with conventional technology and solving the detector-chip interconnection problem as a separate operation. The bonding of connections at a density of one bond per $50 \mu\text{m}$ is not done commercially.

Since we want information from 40 channels per millimeter, the only way to avoid a vast mass of cable is to multiplex locally.

The technology of storing analog signals on MOS capacitors is conventional, so the obvious technologies (probably the only practical ones) are NMOS or CMOS. Stanford University had (and have) NMOS processing and design experience, and were willing to collaborate with us and do the chip design.

What we wanted was a chip with lots of circuits on it at a pitch of less than $50 \mu\text{m}$. Each circuit was required to consist of a low noise amplifier, a set of switches and a storage capacitor to hold the signal while waiting for it to be read out in sequence with many others (Fig. 8).

A design was achieved with the required characteristics -- but only after a long series of iterations with computer studies of the circuit.

Basically NMOS is not a very convenient technology for low noise linear amplifiers, but SPICE enabled the designers to find a solution. SPICE is one of the best known, and most readily available Simulation Programs for Integrated Circuit Electronics. Given as input the components between circuit nodes it will, by stepwise calculation, calculate the response of any (non-linear) circuit to any input wave form. The components may be specified by their known characteristics, or their physical structures (for bipolar transistors, or MOS transistors) will be used to model their characteristics.

For our purpose -- a small circuit -- this simulation was ideal. For large complex circuits other techniques of calculation are necessary since SPICE is far too slow even on large mainframe computers.

The top mask of a (nearly) final version of the circuit is shown in Fig. 9.

The chip works, and is under test. The most remarkable feature of it is in my opinion the fact that there are digital signals and transient swings of volts all over the chip, and yet signals of $\sim 1 \text{ millivolt}$ are visible above noise.

The chip has taken nearly two years to get running, and it seems not to be an exceptionally long time for people outside the major hi-tech companies.

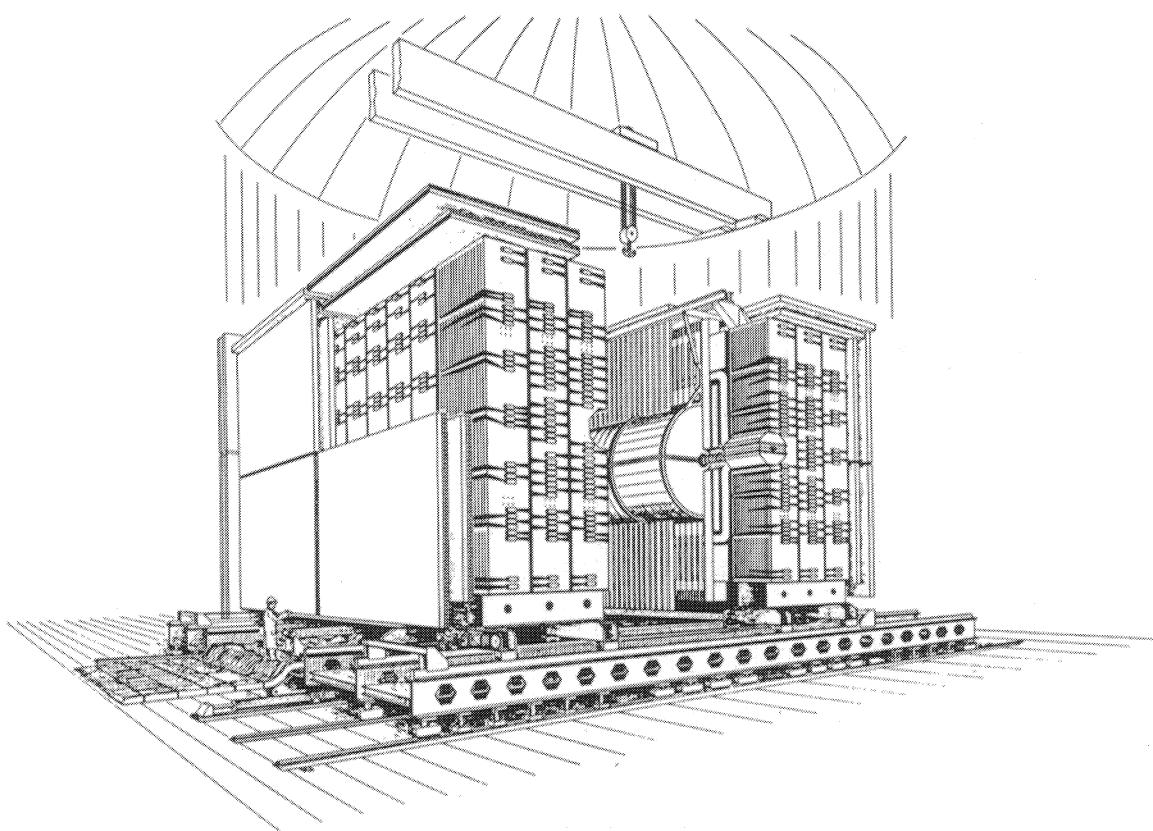


Fig. 1

EVENT 2958. 1279. 69576

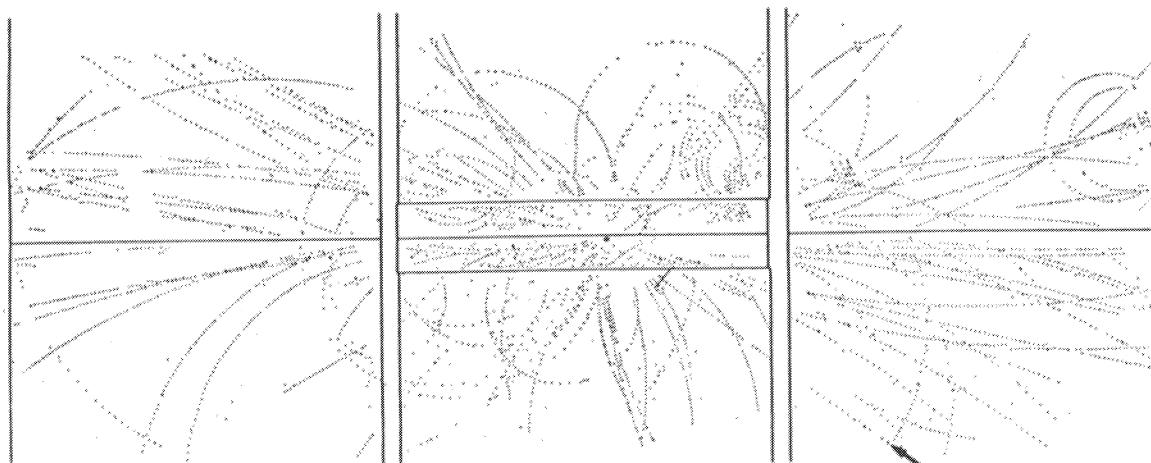


Fig. 2

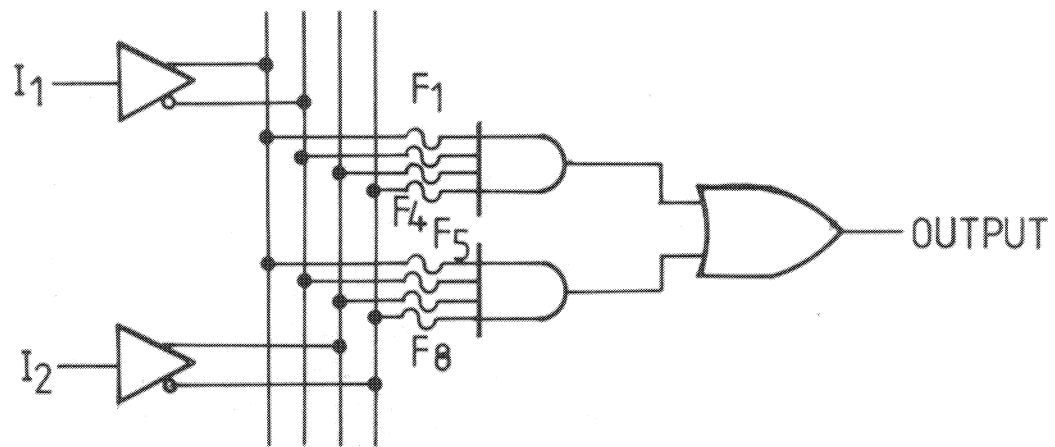


Fig. 3

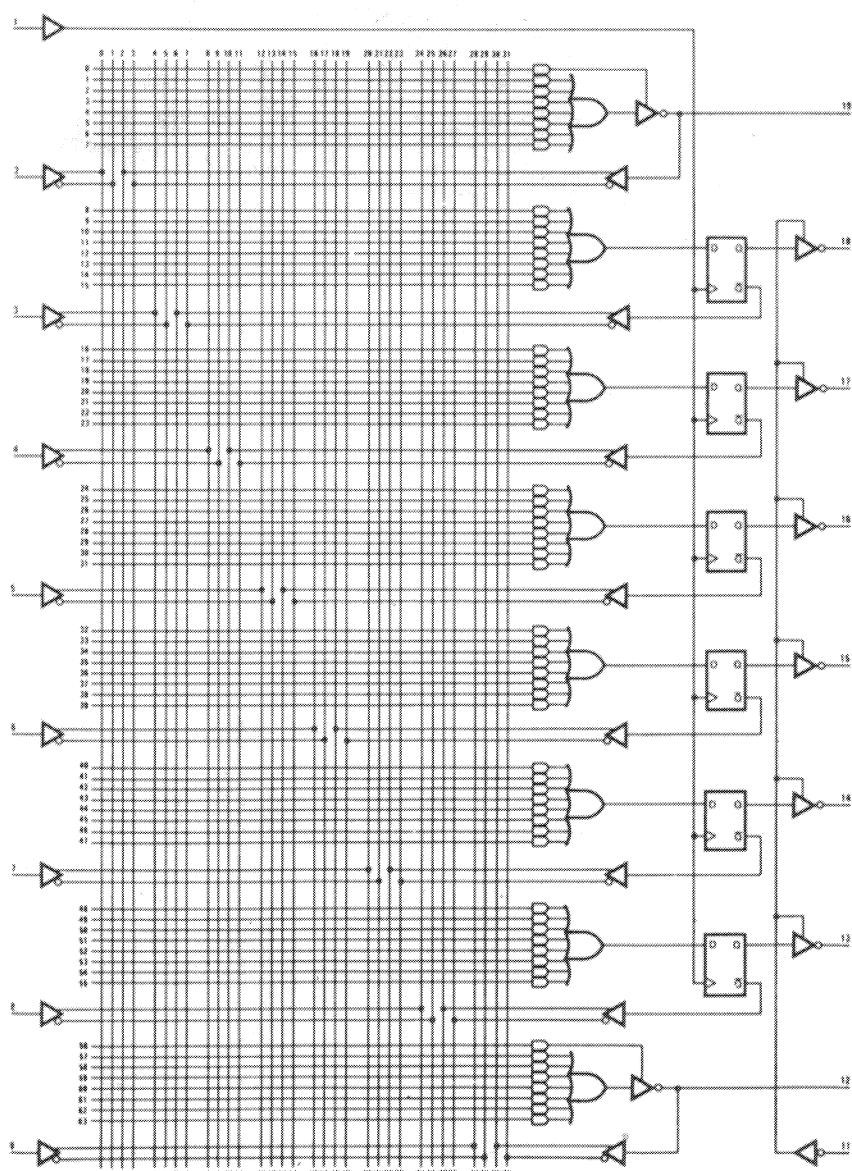


Fig. 4

LINEAR SEMI-CUSTOM DESIGN

Master-Chip

Chip Size: 90 x 90 mils
Total Components: 309
NPN Transistors
 Small Signal: 58
 High Current: 2
PNP Transistors: 18
Schottky Diodes: None
Pinch Resistors
 $30k\Omega$: None
 $60k\Omega$: 8
 $100k\Omega$: None
Bonding Pads: 18
Max. Operating Voltage: 20V
Diffused Resistors
 200Ω : 19
 450Ω : 68
 900Ω : 65
 $1.8k\Omega$: 44
 $3.6k\Omega$: 27
Total Resistance: $269k\Omega$

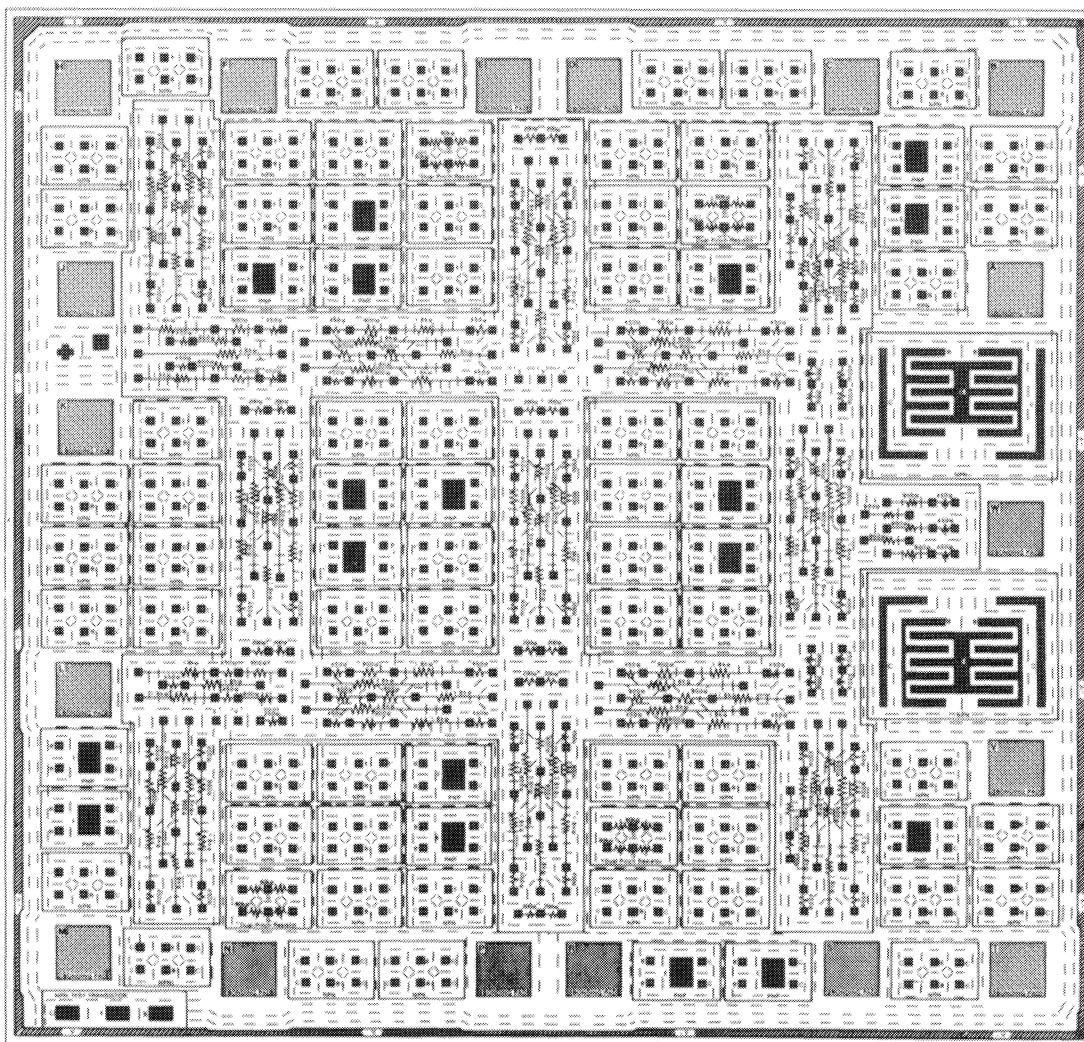


Fig. 5

E = Emitter
B = Base
C = Collector

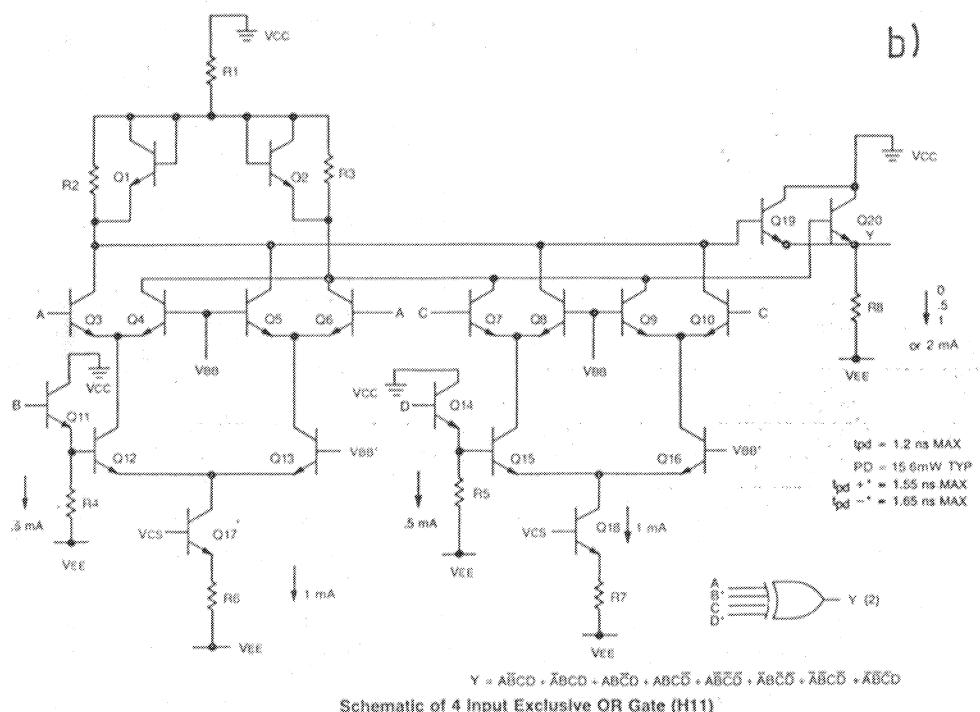
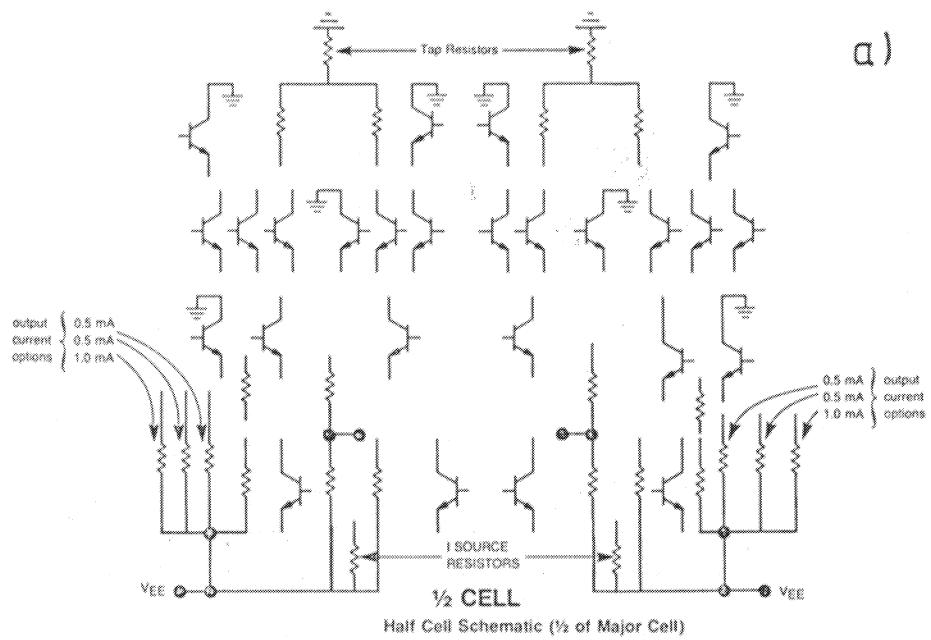


Fig. 6

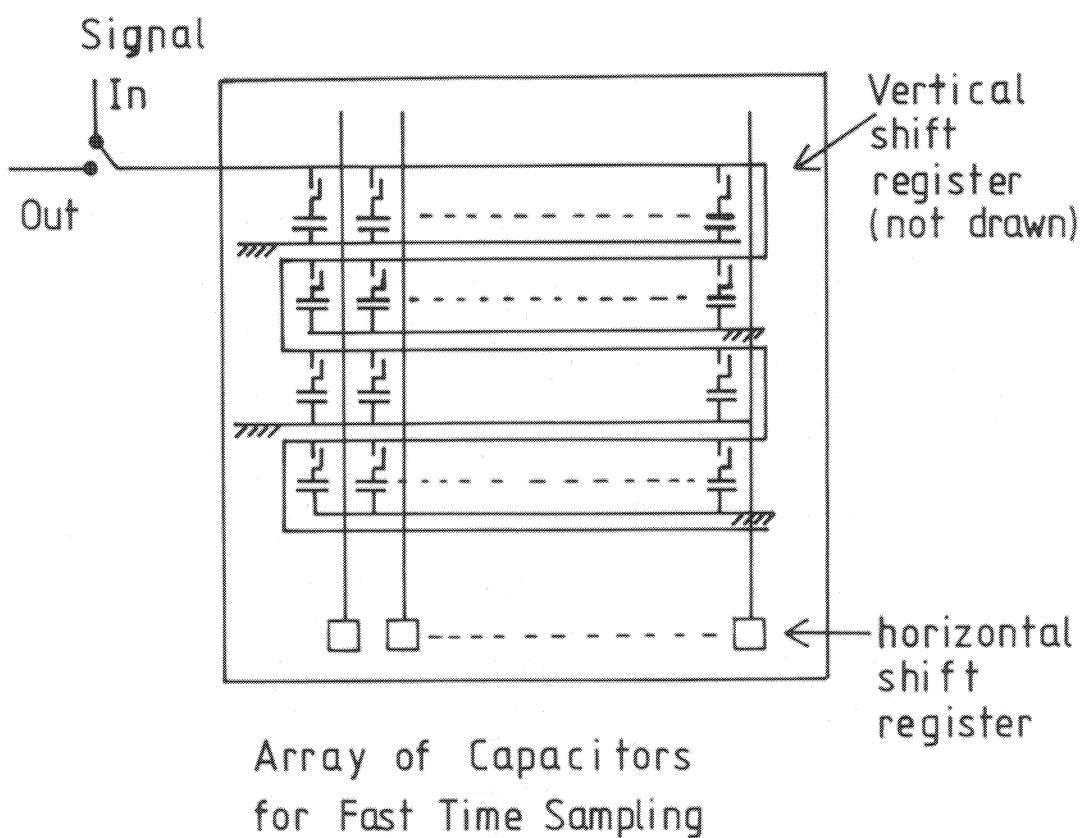
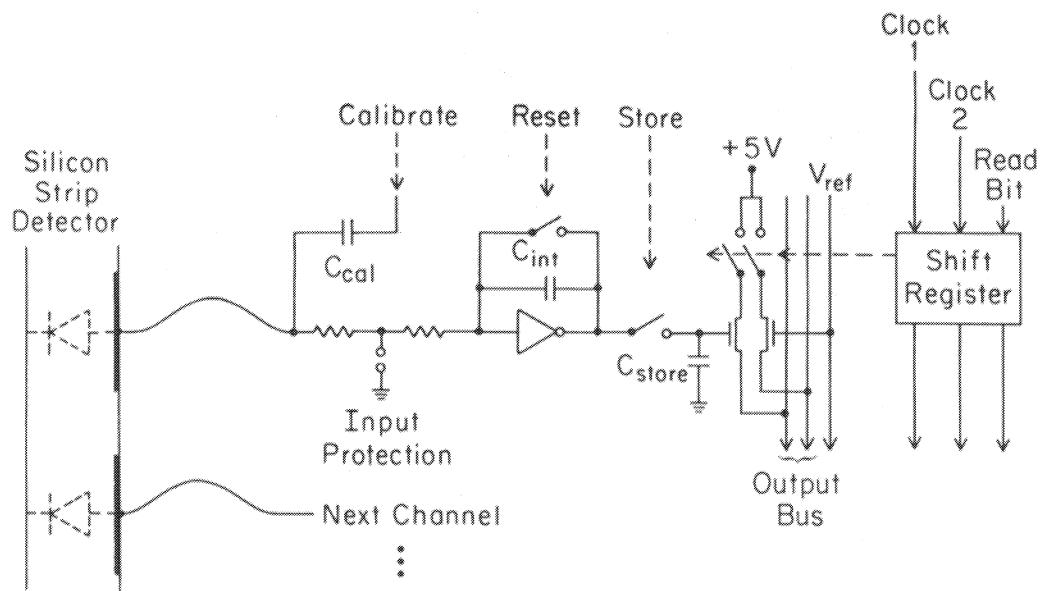


Fig. 7



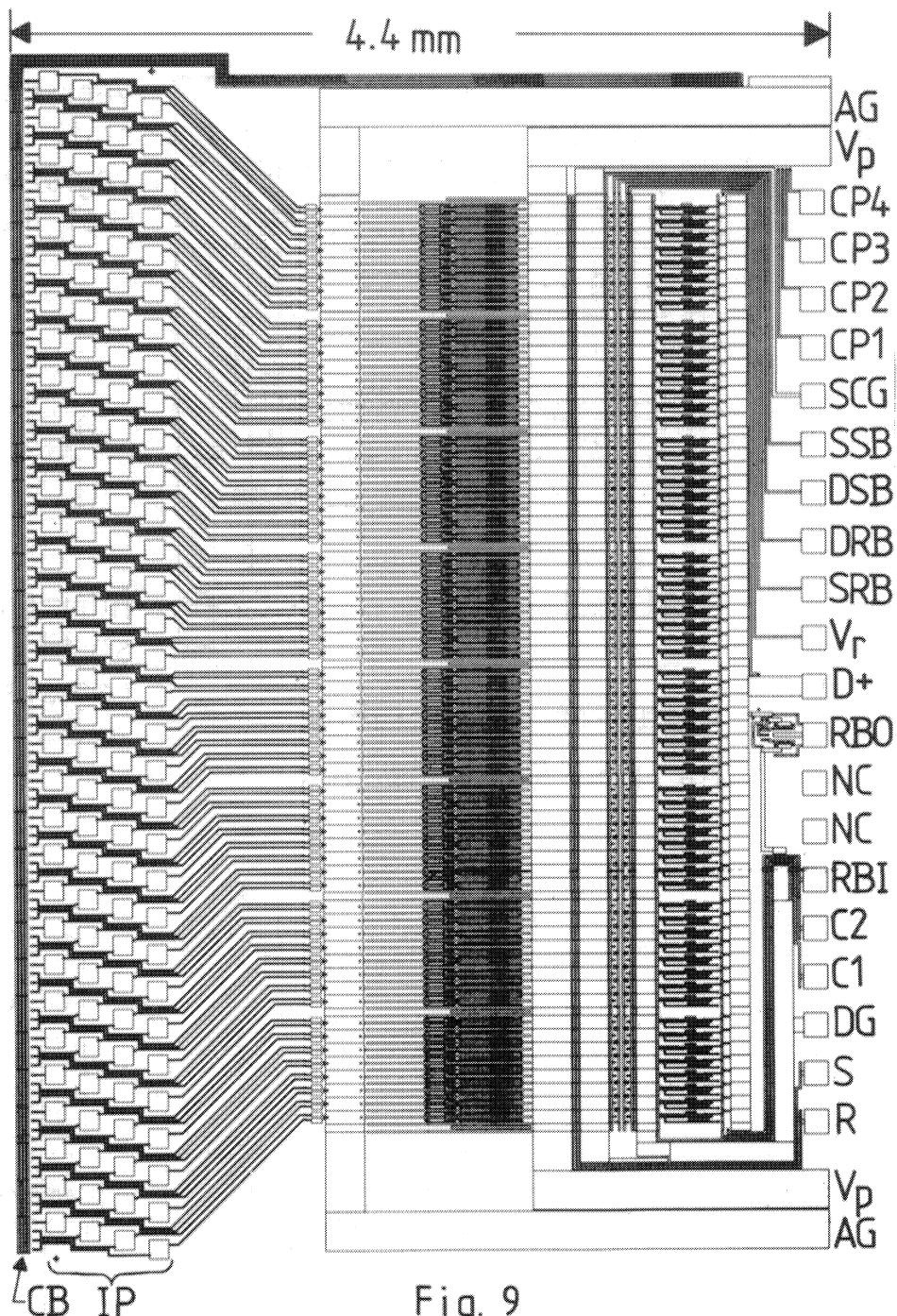


Fig. 9

Enlarged view of chip layout.

AG = Analog Ground
V_p = Analog + 5 pulsed
CP4 = Calibrate Pulse 4
CP3 = Calibrate Pulse 3
CP2 = Calibrate Pulse 2
CP1 = Calibrate Pulse 1
SCG = Storage Cap Ground
SSB = Source, Signal Bus
DSB = Drain, Signal Bus
DRB = Drain, Ref Bus
SRB = Source, Ref Bus
V_r = V_{ref}

D+ = Digital + 5V
RBO = Read Bit Out
NC = No Connection
RBI = Read Bit In
C2 = Clock 2
C1 = Clock 1
DG = Digital Ground
S = Store
R = Reset
IF = Input Pads
CB = Calibrate Buses

MICROPROCESSORS: FROM BASIC CHIPS TO COMPLETE SYSTEMS

R. W. Dobinson,*)

University of Illinois, Urbana, Illinois, USA.

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

* * * *

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

Siegfried Sassoon

April 1917

1. AIMS OF THESE LECTURES

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

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

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

*) Permanent address: CERN, Geneva, Switzerland.

2. SOME NECESSARY HISTORY

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

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

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

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

The genesis of the microprocessor was sparked off by a contract between Busicom, a now defunct Japanese calculator firm, and INTEL, then a young semiconductor company specializing in memory products. Busicom wanted INTEL to produce a chip set of 12 ICs for a family of high-performance programmable calculators. Typical calculator chips in production at that time contained around 600-1000 transistors per chip. INTEL thought they could double this and were led to propose an alternative solution to the 12-IC-chip set proposed by Busicom. They suggested and implemented a three-chip approach; a flexible programmable (micro) processor chip and two memory chips, one for program and one data. Later a fourth chip was added for output expansion. The microprocessor chip was designated the 4004, and in its final form it contained about 2300 transistors. INTEL, having renegotiated their original contract with Busicom, proceeded to sell the MCS-4 chip set to other customers for non-calculator applications, and the microprocessor boom duly

commenced. INTEL had some misgivings about the viability of making the chips generally available, apparently on the grounds that it might not be economical or possible to compete with then existing minicomputers. The 'genius' behind the decision to go ahead was based on a realization that the microprocessor potential was not as a minicomputer replacement but as a way of bringing intelligence to many new areas.

The INTEL 4004 chip is thus the grandfather of all microprocessors. It was first advertised to the world in November 1971 in an issue of Electronics News that also announced another significant event: the invention of an 'erasable and re-programmable Read Only Memory'. The 4004 was a 4-bit microprocessor. It manipulated data 4 bits at a time. There were 46 instructions (typical execution time 10 μ s) and up to 4 k separate memory locations that could be directly addressed. Within a rather short time some of the shortcomings of the 4004 (for example it had no interrupt capability) were improved on by the INTEL 4040. This and other 4-bit microprocessors, notably from Rockwell and Texas Instruments, entered a large number of applications areas. However, the introduction of new designs in the 4-bit field would seem to be unlikely, except perhaps for high-volume specialized applications; 8- and 16-bit microprocessors are now the norm. Rather soon after the advent of the first 4-bit chips, more powerful 8-bit microprocessors became available at only slightly higher prices.

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

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

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

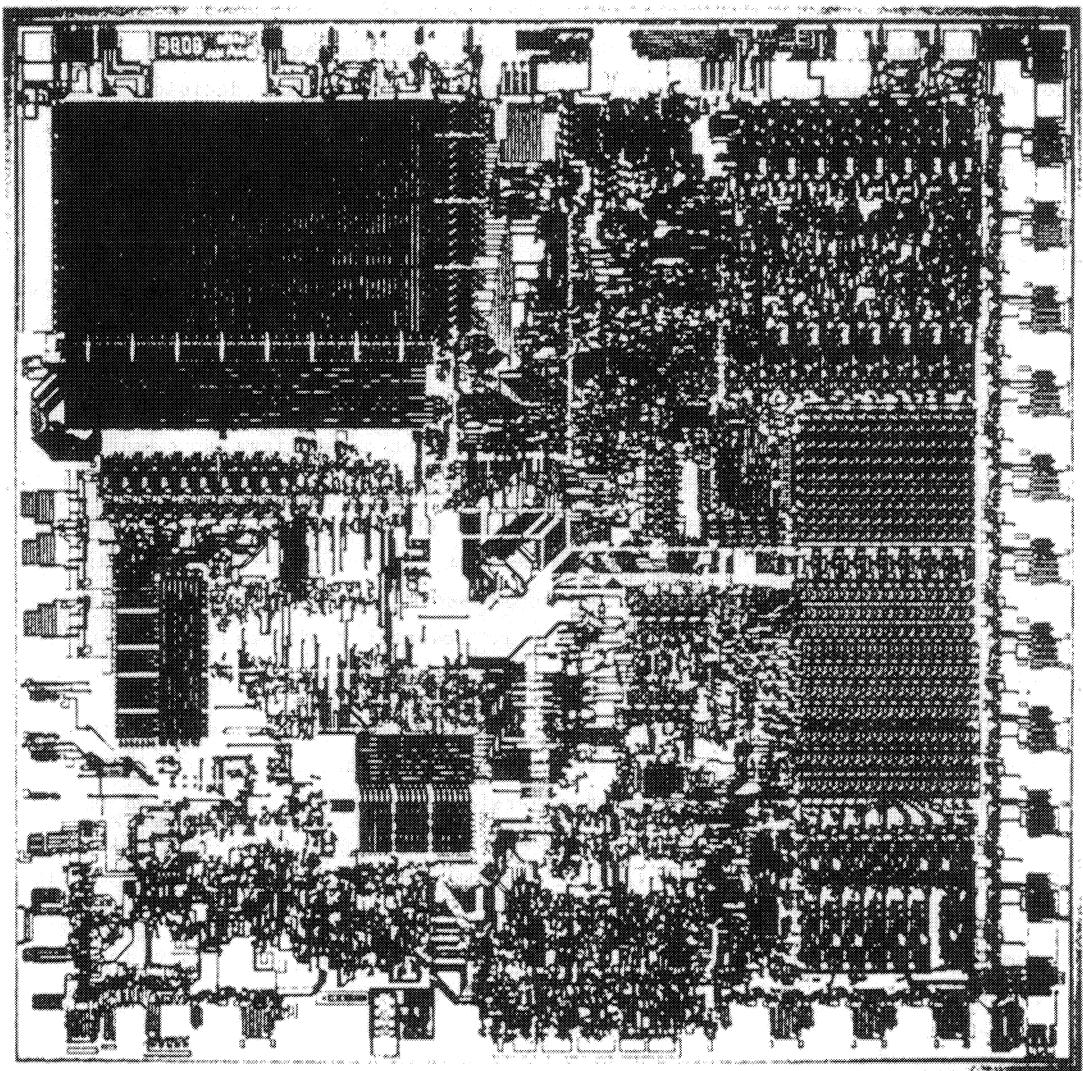


Fig. 1 The INTEL 8086

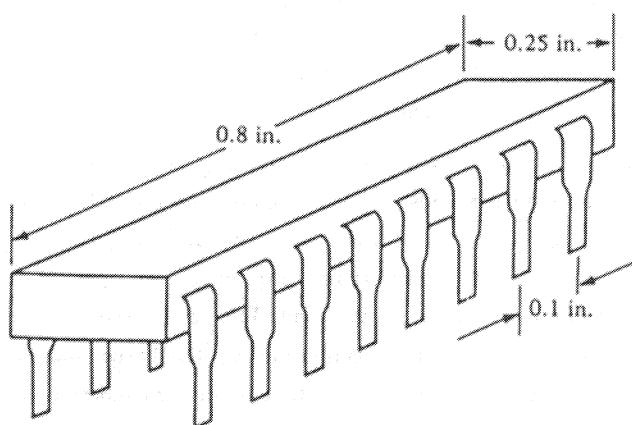


Fig. 2 Dual in-line package (DIP) for an integrated circuit

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

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

3. MICROPROCESSOR BASICS

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

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

- a) Arithmetic instructions: add, subtract, multiply, divide, increment, decrement, negate, complement.
- b) Logical instructions: AND, OR, exclusive OR, bit manipulation and testing.
- c) Information transfers: information is moved from one location to another without being changed. Sources and destinations for the information can be CPU internal registers, memory, and I/O devices.
- d) Branches in normal sequential program flow: unconditional jumps, conditional jumps, subroutine calls, subroutine returns, software interrupts.
- e) CPU control: no operation, halt, enable/disable interrupts.

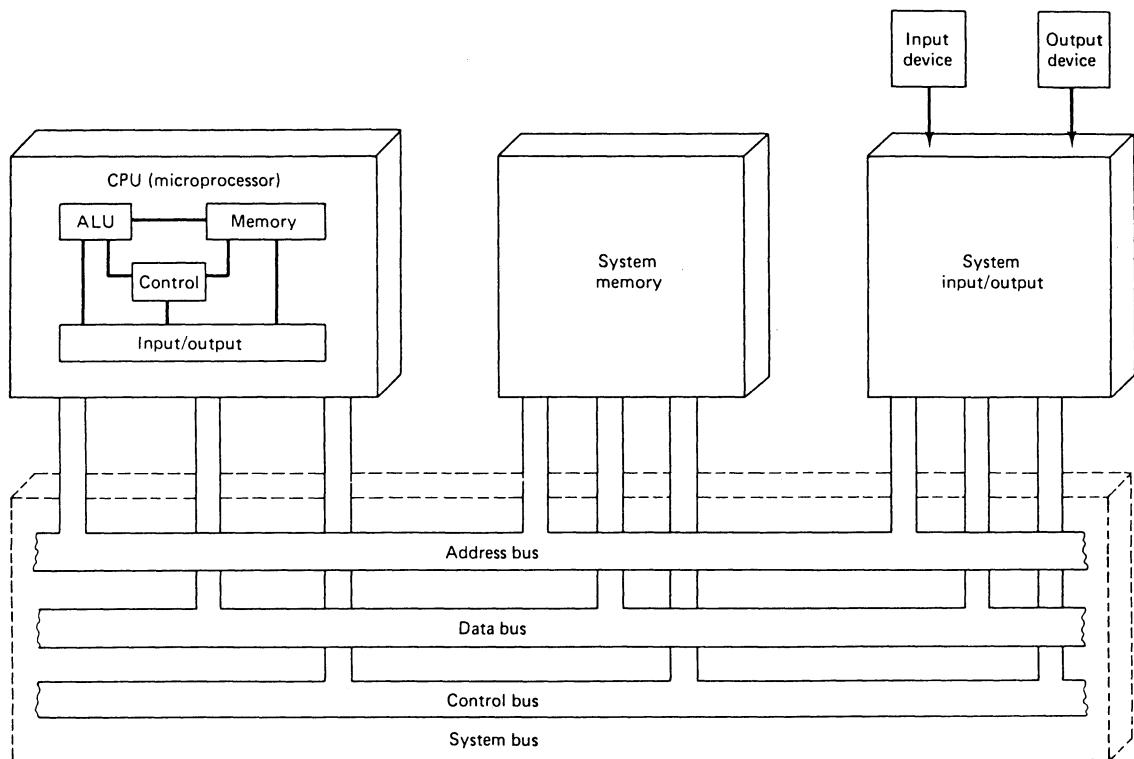


Fig. 3 The component parts of a basic computer

The number of operands specified for any opcode can vary from none to several. Normally 'several' in microprocessors means up to two: a source and a destination for a move; two sources of data for an addition with the results going to overwrite one of the original sources. Operands are specified in terms of addresses of memory locations, internal CPU registers, and I/O ports. The final effective addresses to be used in accessing information can be specified directly as part of the instruction or indirectly via pointers. For example, an address specified in an instruction may point to another address which contains information (data) to be operated on. A still more complex way of specifying an effective address for an operand is indexing; the effective address is the sum of a special register in the CPU and an offset specified as part of the instruction. We see in general that the effective address of an operand is the result of some address computation. Later we will come back to how instructions are fetched, decoded, and executed. Let us now turn to a more detailed picture of how a computer system is put together, in particular the internals of a microprocessor chip.

Figures 4a and 4b show a typical microprocessor-based system. Part (a) of the figure represents the functions normally incorporated in a 'standard' microprocessor chip, the CPU. The registers shown are representative rather than definitive. Part (b) shows various functions that can be added to complete the system. They are in a 'standard system' provided as separate chips. This having been said, we remark that since 1976 it has been possible to incorporate some of these functions on the processor chip itself. One has traditionally referred to the resulting device as a single chip microcomputer. A

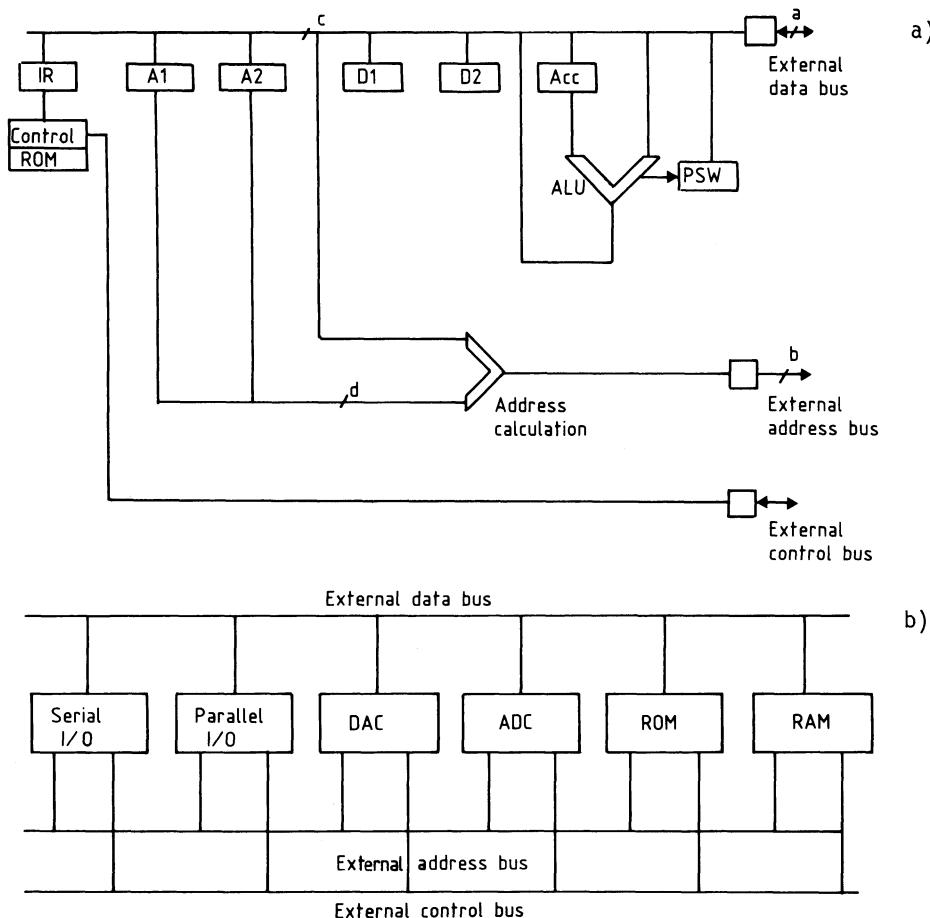


Fig. 4 a) The functions normally incorporated in a standard microprocessor chip: a, b, c, and d are the widths of the buses; A1 and A2 are address registers; D1 and D2 are data registers; Acc is the accumulator; IR is the instruction register; PSW is the processor status word.

b) Additional functions added to a) to make up a standard system.

typical microcomputer chip might contain a CPU, a memory, and an analog-to-digital converter (ADC) -- nearly all the features needed in a small computing system for use in simple control applications in many fields, from car ignition systems to domestic appliances. A clear distinction between microprocessors and microcomputers seems to be disappearing as manufacturers routinely squeeze more than just a bare CPU on most 'microprocessor' chips (see Fig. 5).

Let us now go in turn through the different components of a standard microprocessor chip. A single internal data bus is shown as providing a communications path between the different registers and the ALU. Operands are brought from registers and memory to the ALU, and afterwards the results are returned to registers or memory via the same bus. Often one of the registers, the accumulator(s), plays a special role -- it provides one of the inputs to the ALU and results are returned to it. Note that the use of a single bus requires the least space on a chip. As space is a critical factor when designing and producing a chip, most microprocessors have a single-bus architecture. The disadvantage of this is slower operation. The bus is multiplexed; it is used for different purposes at different times. Improved execution speed would result if dual or

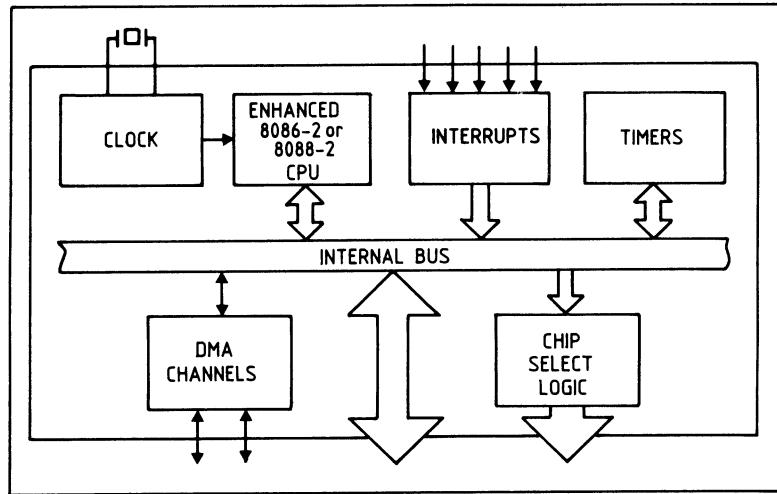


Fig. 5 The INTEL 186 chip. A powerful processor and support functions on a single chip.

even triple buses were used (see Fig. 6). In the case of a dual bus system, a source bus connects to both inputs of the ALU and a separate destination bus is provided. A triple-bus system provides separate source buses for each input of the ALU and an extra destination bus.

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

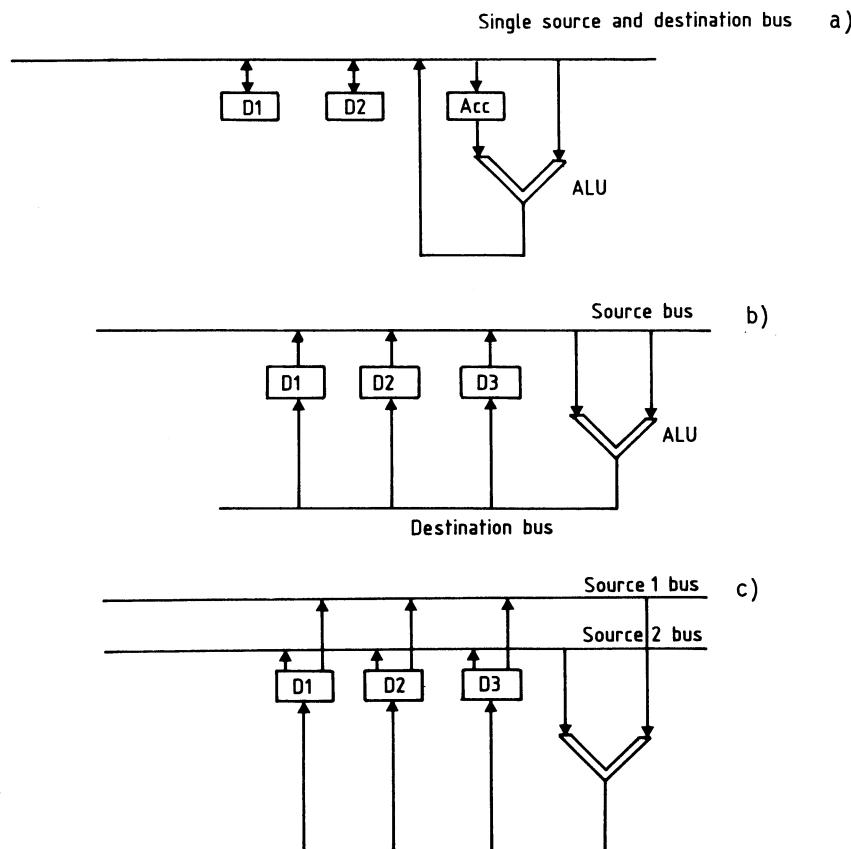


Fig. 6 a) Single-bus architecture; b) Double-bus architecture;
c) Triple-bus architecture.

internal data registers and buses. Execution speed also depends on the type of semiconductor technology used.

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

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

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

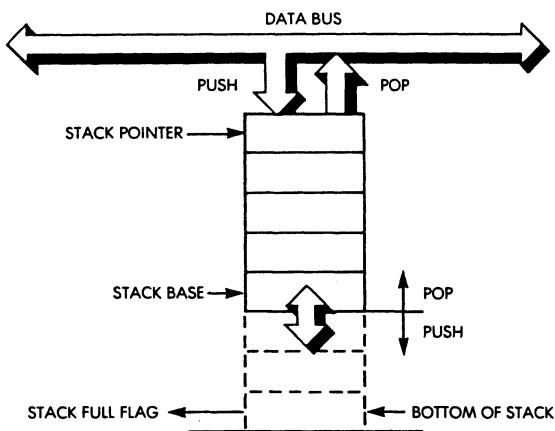


Fig. 7 Implementation of a stack

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

- 1) It improves utilization of the limited chip area by providing more logic per square millimetre than a random logic design. This is because it is easier to build regular structures such as memories in a compact way.
- 2) The microcode burned into the ROM can be changed relatively easily by the manufacturer. This is a major advantage in correcting early design mistakes or for adding instructions and/or improvements at a later date. It is even possible to emulate an entirely different instruction set for the microprocessor chip. For example, a Motorola 68000 has been microprogrammed to emulate part of the IBM series 370 instruction set. Some minicomputers allow end users access to microcode on their machines. If you have a spare year or so you can amuse yourself adding your own instructions. At the microprocessor chip level the end user does not get to touch the internal microcode--not unless your name is IBM.

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

We are now in a position to see how an instruction is executed by our microprocessor chip. Be warned that what I am going to say is very much simplified. In addition, different processor chips work in different, sometimes more complicated, ways. What I will present does however follow fairly closely what happens in an INTEL 8080.

The execution of every instruction starts with an instruction fetch phase. The starting address of the next instruction to be executed is contained in the program counter whose contents are gated out onto the external address bus together with appropriate control signals during an external memory access cycle. The contents of that memory location are placed in the instruction register (IR) and the program counter is incremented to point to the next location. The opcode contained in the IR is used to trigger a sequence of corresponding micro-operations. Each step in the sequence corresponds to an internal state. Each state is associated with the execution of a microinstruction and we assume lasts one tick of the master clock. Depending on the instruction being executed, further external memory cycles may be required to fetch source operands associated with that instruction. These operands may be contained in successive memory locations, in which case they are pointed to by the program counter, or they may be accessed via other address registers. The required operation can now be performed using the ALU. If an external memory location has been specified as a destination operand in the instruction, a memory cycle is required to store the results of the operation.

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

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

4. MICROPROCESSOR BUSES

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

The first obvious function of the bus is to transfer information into and out of the chip, to and from memory and I/O devices. In a basic transaction the processor asserts an address, and data are either written to or read from that address. Control lines are used to command the required direction of data transfer and to synchronize or time the whole operation. Figure 9 summarizes these basic ideas.

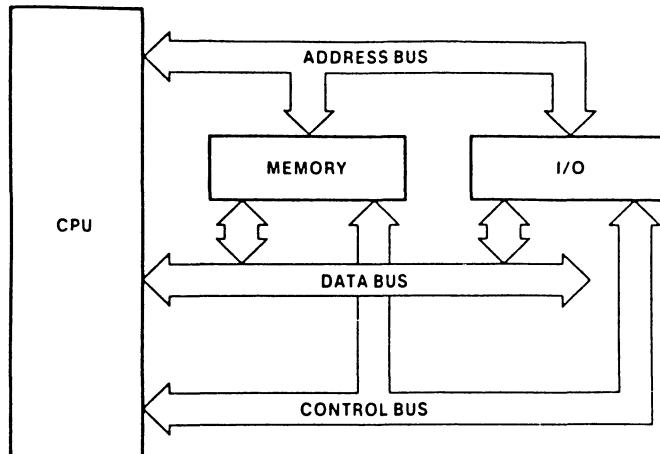


Fig. 8 The component level bus of a microprocessor chip

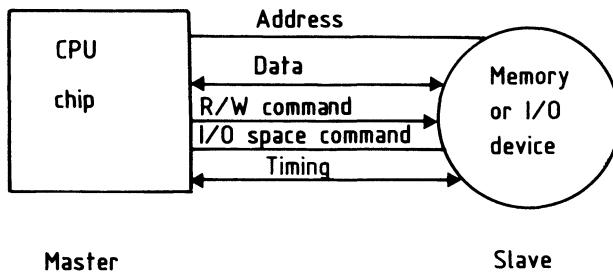


Fig. 9 Basic transaction between a microprocessor chip and memory or I/O device.

The address information provided by present-day processors typically comprises between 16 and 32 parallel output lines from the chip. These lines provide a unique identification for between 65536 and 4295×10^6 different locations. Some microprocessors make a distinction between how memory and I/O devices are addressed. They do this by providing a second I/O address space. This space, which is normally considerably smaller than the main address space, could be selected for example via extra control bus lines. Note that Fig. 9 shows an I/O address space command line. Special I/O instructions are available for accessing devices located in I/O space. Processors which do not provide a separate I/O space assume that you will allocate both memory and I/O device addresses from the single main address space. You can, if you so choose, ignore I/O space even when the chip provides it for you. Treating memory and I/O device locations in the same way is sometimes an advantage. The I/O operations are then referred to as memory mapped. However, there can also be advantages in keeping the two types of operation separate; when there are only 16 bits of main memory address information, a user may be reluctant to sacrifice part of the space unnecessary for I/O -- in some cases DMA transfers between memory and I/O are more convenient if distinct I/O addressing is used.

The data-bus width is slightly more variable in size than that of the address bus. It ranges from one byte wide on the low end and some medium performance chips, through 16 bits for the INTEL 8080, 186, 286, and earlier Motorola 68000 series, to 32 bits for the latest Motorola 68000 offering, the 68020.

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

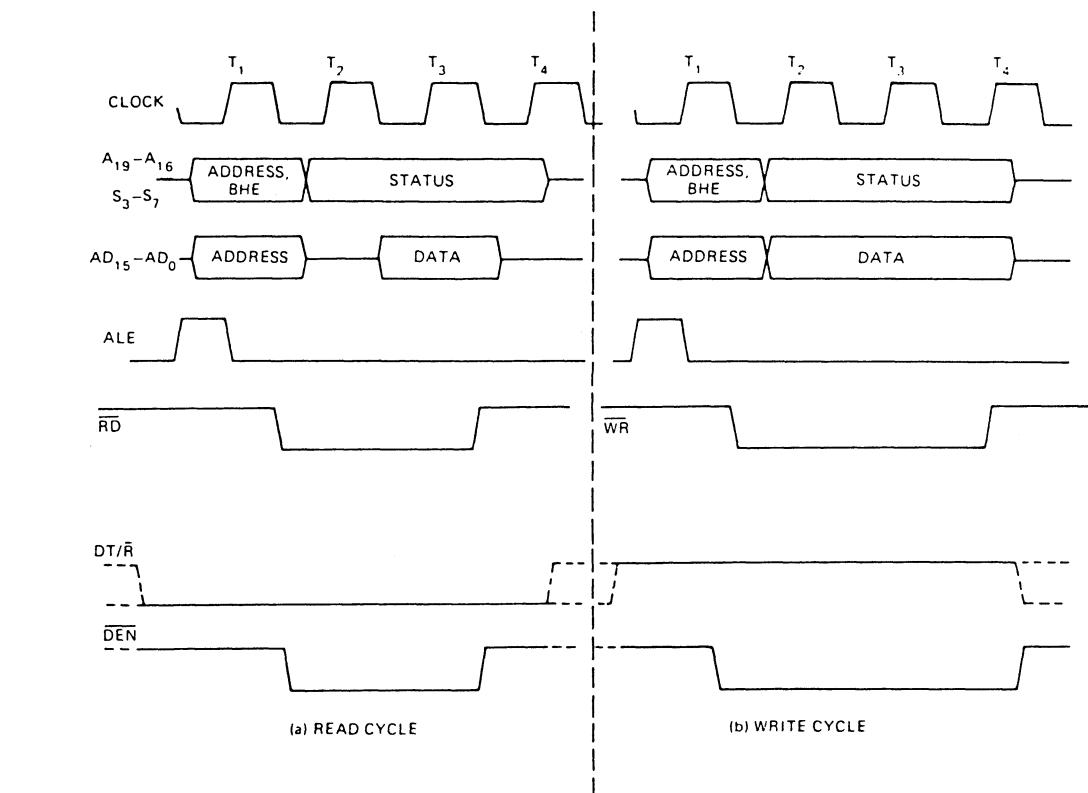


Fig. 10 Use of the ALE line on an INTEL 8086/88 to demultiplex address and data. The address, which appears briefly at the start of a transaction, can be latched on the low-going edge of ALE. Note that the high address lines are used to provide status information later in the transaction, and the low sixteen address lines are used for data transfers.

allows the designer to work with separate address and data lines thereafter. Even with the removal of the 40-pin restriction there still seem to be insufficient pins to go round, and many current (and even new) microprocessor and support chips use multiplexing for getting signals into and out of a chip. We note one final point in passing: dual in-line packaging is rapidly being replaced by other 'alternatives' when large numbers of pins are involved.

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

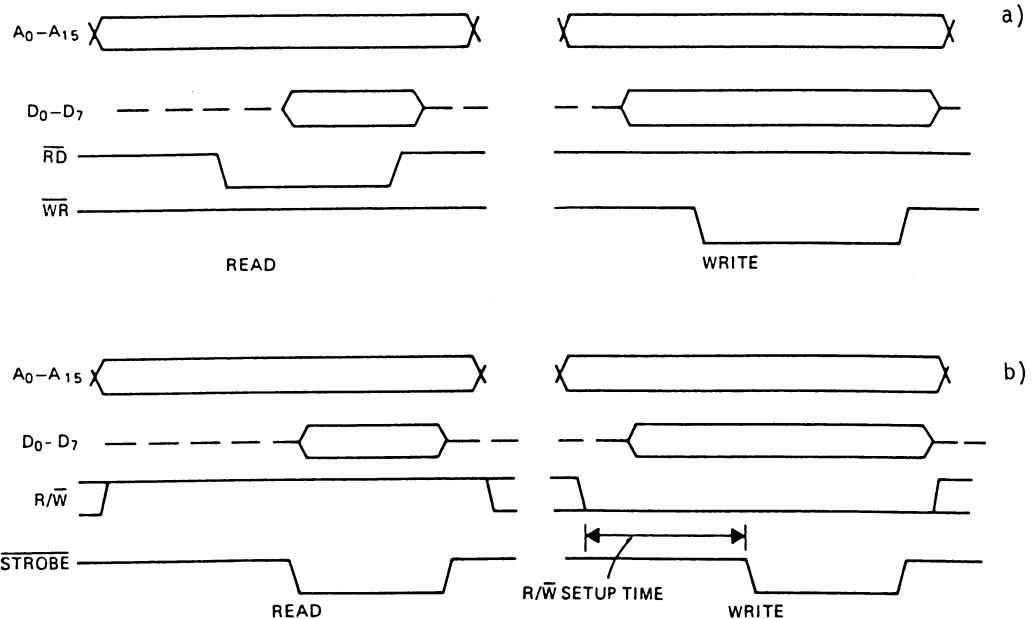


Fig. 11 Comparison between the timing of a) an 'INTEL 8080 type' bus with b) a 'Motorola 6800 type' bus

is that if slow slaves are to be accommodated then the processor must be slowed down to a speed acceptable to the slowest slave. This reduces the speed of all transactions -- particularly annoying if a slow device is only rarely addressed.

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

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

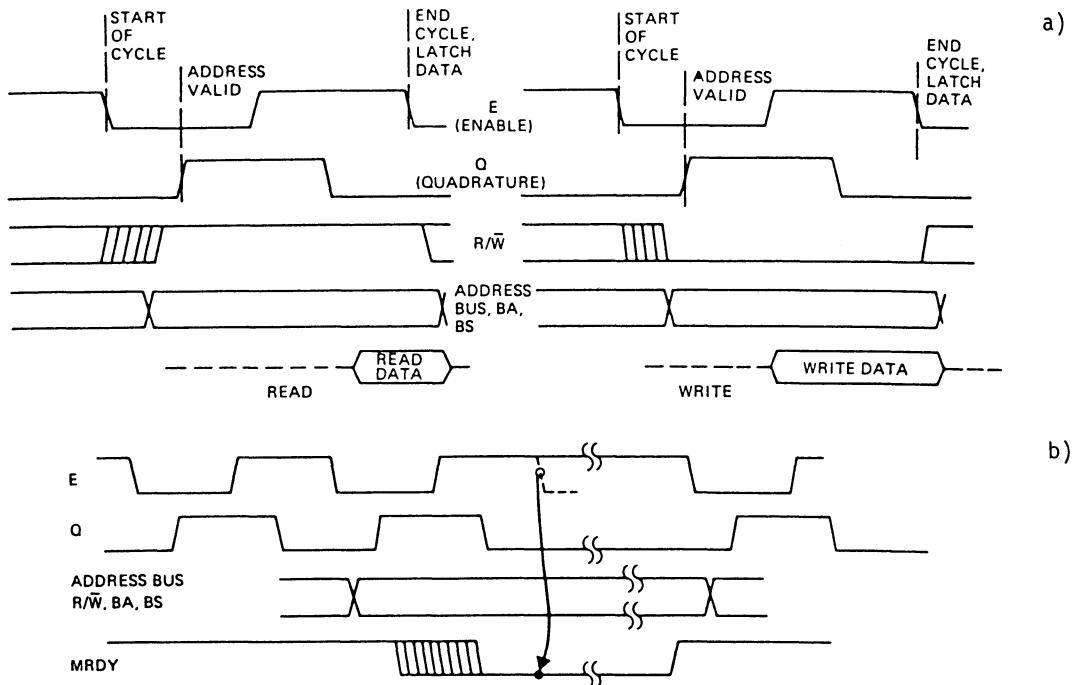


Fig. 12 Timing for the Motorola 6809: a) shows how Reads and Writes are normally carried out; b) shows how a slow slave can stretch bus cycles.

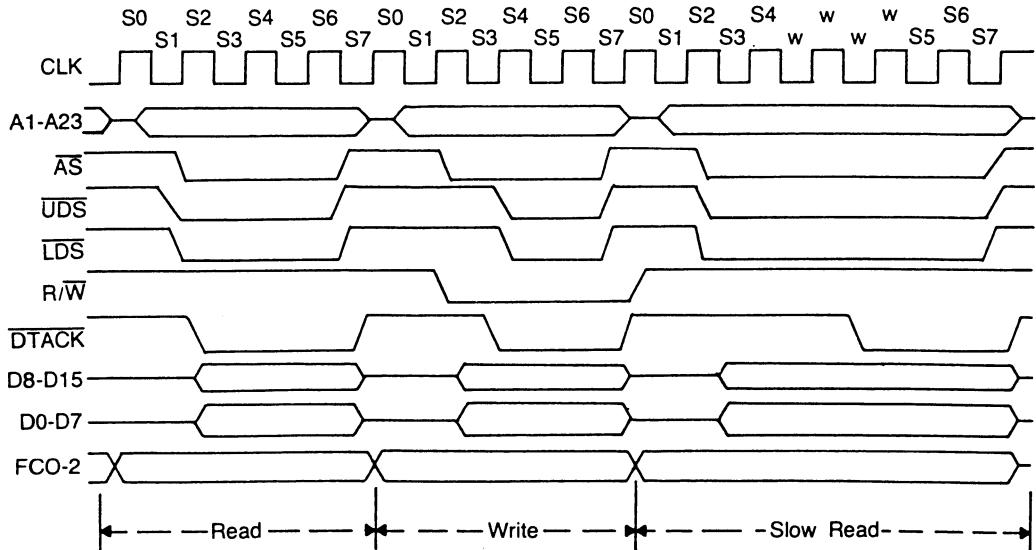


Fig. 13 Asynchronous timing of Motorola 68000

being transferred on the upper, the lower, or both bytes of a 16-bit word, and the Read/Write line gives the direction of transfer. The data acknowledge (\overline{DACK}) input from a slave to the processor is a handshake signal. Until it is asserted -- pulled low -- the 68000 will insert so-called wait states. You can see this clearly in the slow read cycle shown in Fig. 13. Also shown in the figure are 16 data lines and the FC function code lines which identify the type of bus cycle being performed. One of the interesting points about the 68000 is that along with its basic asynchronous mode it supports an alternative synchronous timing mode compatible with the 6800 family. This allows 6800 support chips to be used with the 68000.

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

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

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

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

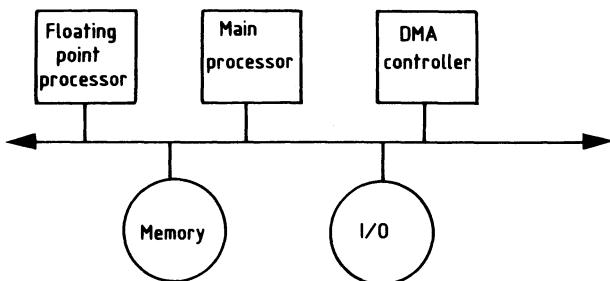


Fig. 14 Sharing of access to memory and I/O by multiple masters: main processor, floating-point processor, and direct memory access controller.

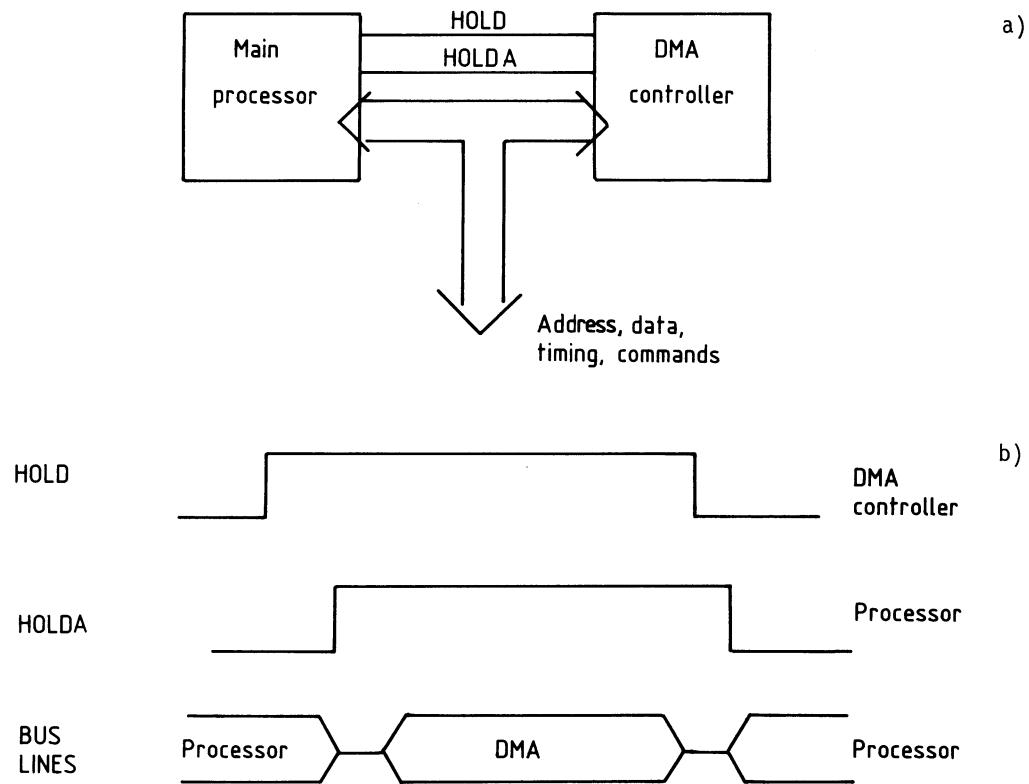


Fig. 15 Use of the arbitration lines HOLD/HOLDA to share the microprocessor bus between the main processor and a DMA controller.

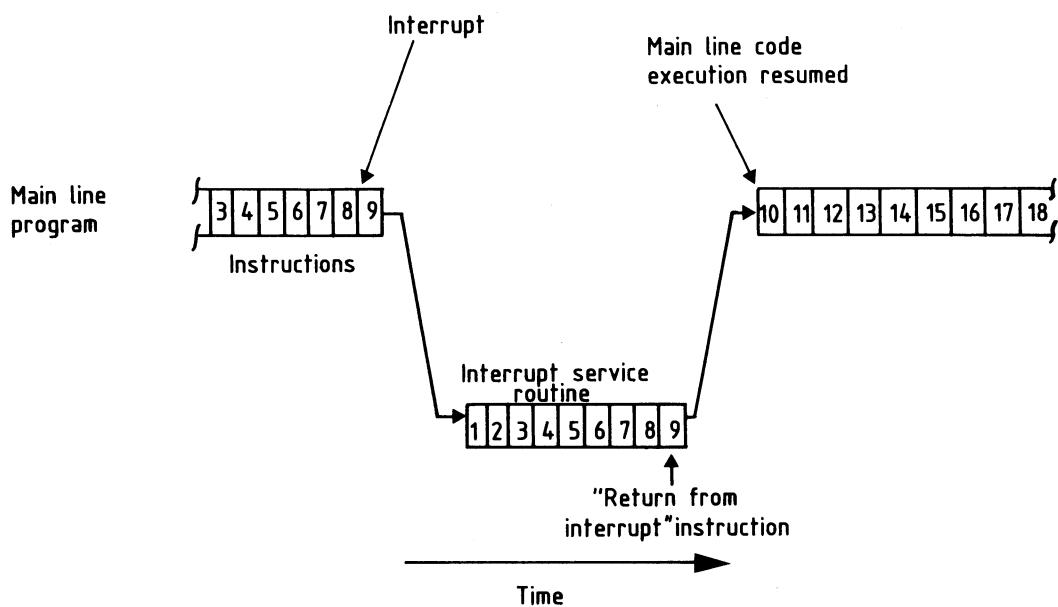


Fig. 16 Basic principles of interrupt handling

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

Figure 17 shows the use of an interrupt controller chip in association with a microprocessor. The interrupt controller is used to handle up to eight separate interrupt

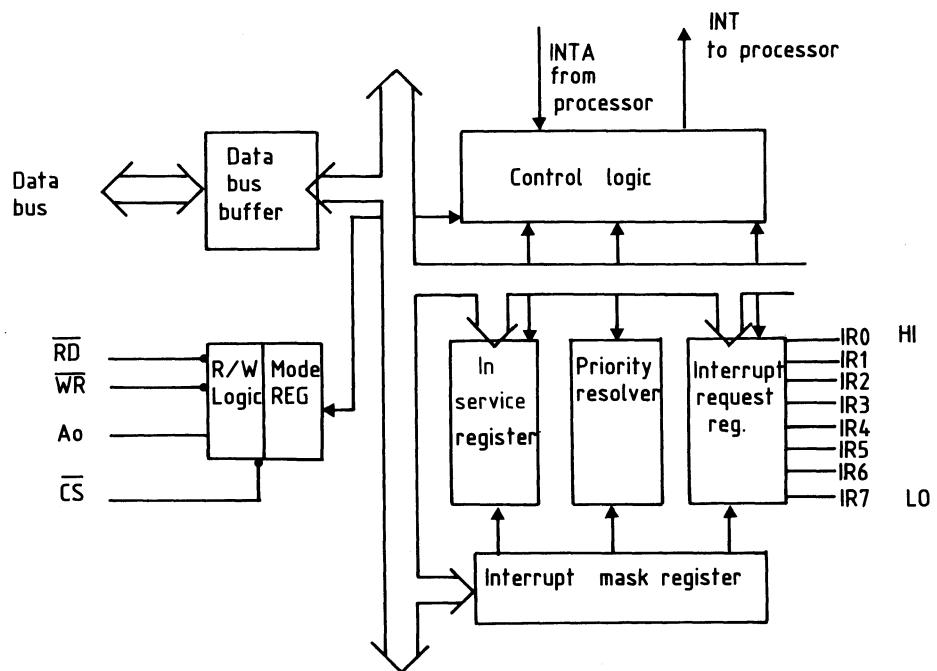


Fig. 17 Interrupt controller. There are eight separate interrupt lines $IR_0 \rightarrow IR_7$. The controller may be programmed internally via the data bus, chip select, and address line A_0 , and the Read and Write strobes RD and WR . The INT interrupt line goes to the microprocessor chip, and in response to the $INTA$ (interrupt acknowledge) a vector is placed on the data bus.

lines from I/O devices. It can be programmed from the processor to enable or disable any combination of the eight interrupt request lines by means of a mask register. The interrupt controller decides which of several simultaneous device requests is serviced first according to several programmable schemes; for example, fixed priority or rotating (round robin) priority. There is an additional priority feature which inhibits any interrupt with lower priority than specified in the 'in service register'. The controller chip will pass on an interrupt request to the 8088 and will respond with the vector corresponding to the device request being dealt with when asked to do so by the processor. A vector can be programmed for each of the eight device interrupts. These vectors point to corresponding entries in the interrupt service table.

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

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

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

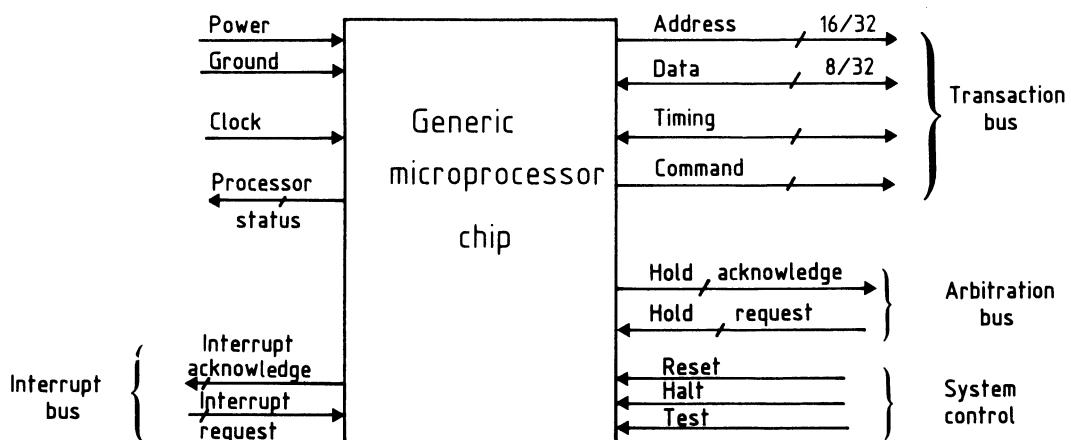


Fig. 18 A generic microprocessor chip

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

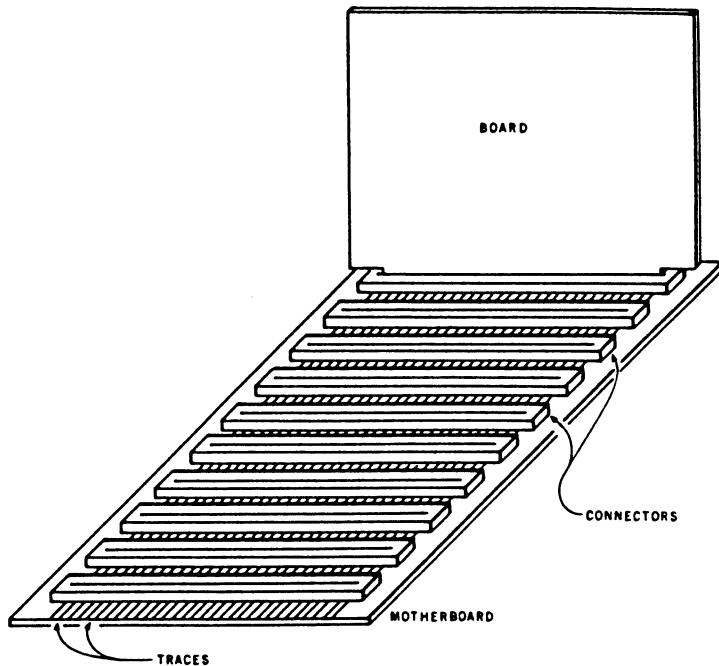


Fig. 19 A backplane bus

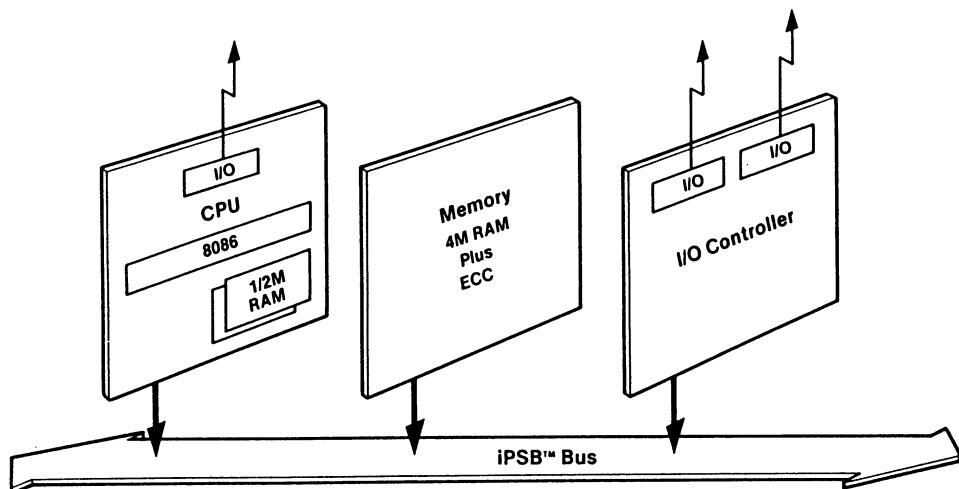


Fig. 20 Synthesis of a basic computing system by CPU, memory and I/O printed-circuits boards.

- When synthesizing a microprocessor-based computing system it is very convenient to adopt a modular approach for both hardware and software. In particular, at a hardware level one would like to be able to purchase printed-circuit boards from the (different) manufacturers who offer the optimum products for tailoring a system to specific needs. It is an advantage to not be tied too closely to a particular processor, memory type, or I/O device, but to be able to evolve in the light of new technologies and end user needs. Defining a standard backplane bus which is processor- and manufacturer-independent and endorsed by a reputable national or international body is a very positive step towards this goal. Many of you in the high-energy physics world will be familiar with the advantages of standardization in a variety of areas including CAMAC.
- The second reason for not using microprocessor-component-level bus signals on a backplane is that in most cases they are just not very suitable for direct use in general purpose multiprocessor (non-hierarchical) systems. The facilities they provide for interruption and arbitration in particular are somewhat rudimentary and require enhancement before they can be considered adequate for many applications.

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

5. THE ROLE OF TTL 'GLUE'

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

Basic microprocessor systems may consist of just a few complex chips. However, more usually a fair number of simpler chips are used in auxiliary but nevertheless very necessary roles. Let us look at some examples.

1. A microprocessor chip may lack the electrical capability to permit its direct connection to a large array of memory chips. It may be necessary to buffer most lines with line drivers.
2. Additional chips or simple latches may be required to demultiplex address and data lines coming out from the microprocessor.
3. Memory chips and other devices for I/O which are attached to a microprocessor bus must recognize when they are addressed and respond appropriately by gating data to and from the bus. Address decoders, multiplexors, as well as basic logic gates, all find a role here.

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

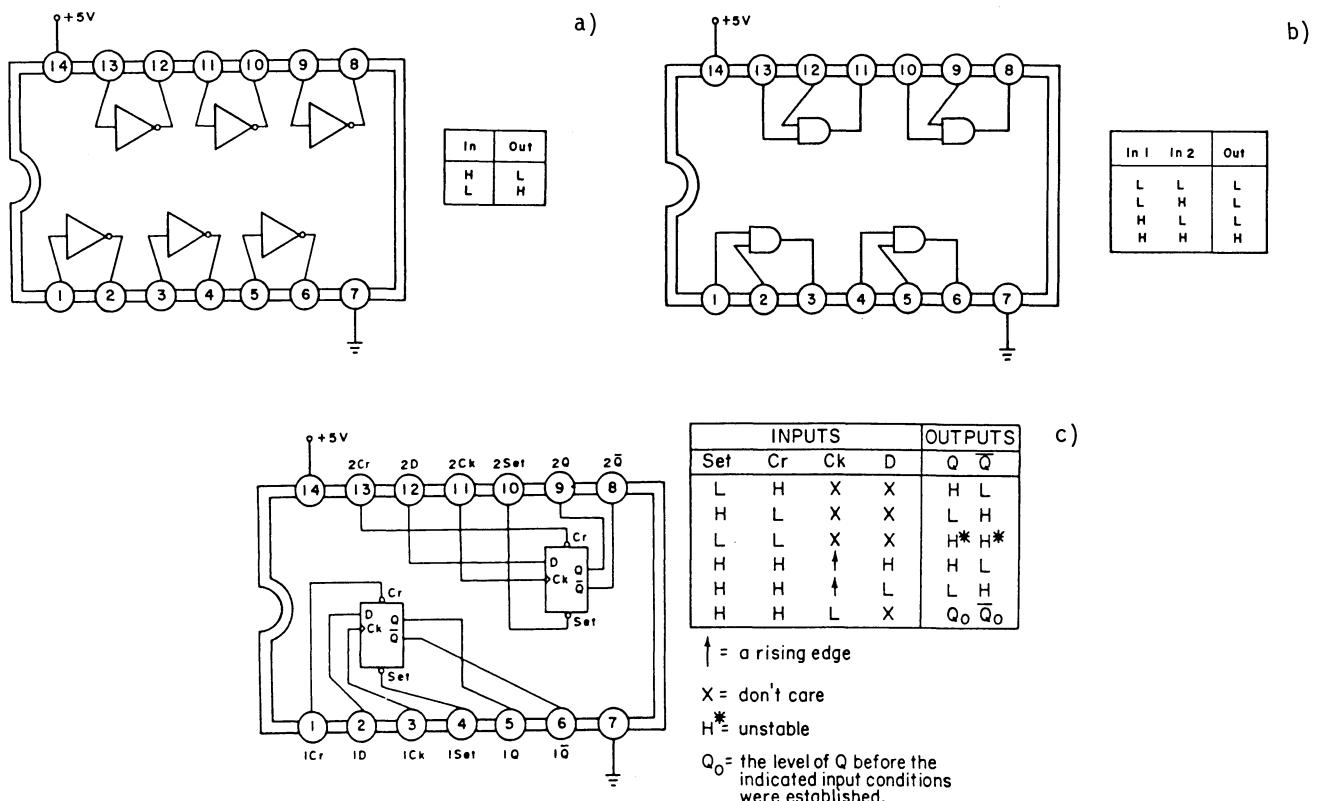


Fig. 21 Examples of TTL glue. a) Inverter; b) AND gate; c) D-type positive edge triggered flip-flop.

6. PRIMARY MEMORY

Memory implies storage of information, and storage functions exist throughout microprocessor systems. We have seen that internally within a chip there exist data and address registers which are used to store and manipulate information. The role of Read Only Memory in storing the microcode used to implement a processor's instruction set has also been mentioned. Two more general types of external storage can be identified: i) Primary memory, which is used to store information that a CPU needs 'immediately'. By immediately we mean with an access time of 100 → 1000 ns. Access time is the time interval between information being requested and when it is available. When we talk about memory we normally mean primary memory. We will discuss the different forms of primary memory later in this section. ii) Secondary memory (or secondary or auxiliary storage) is taken to mean devices such as magnetic tapes and disks. Here, access times are much longer than primary memory but the storage density is much larger. We shall discuss one example of secondary storage later.

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

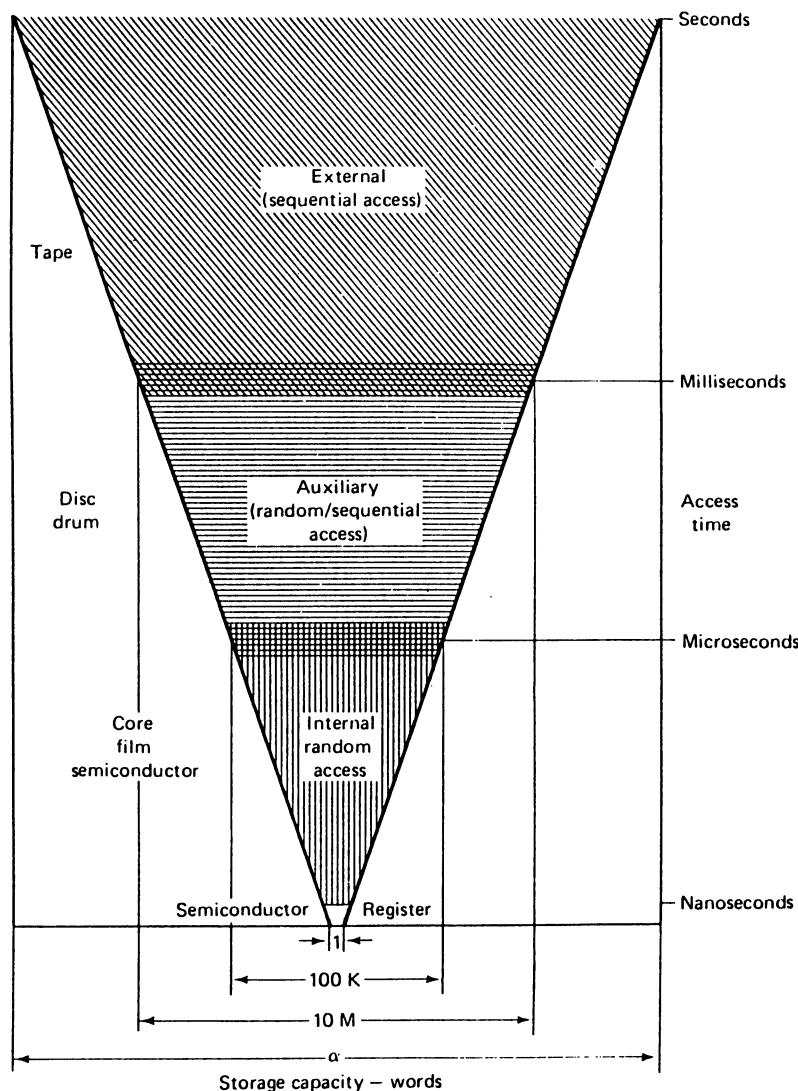


Fig. 22 Storage hierarchy

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

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

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

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

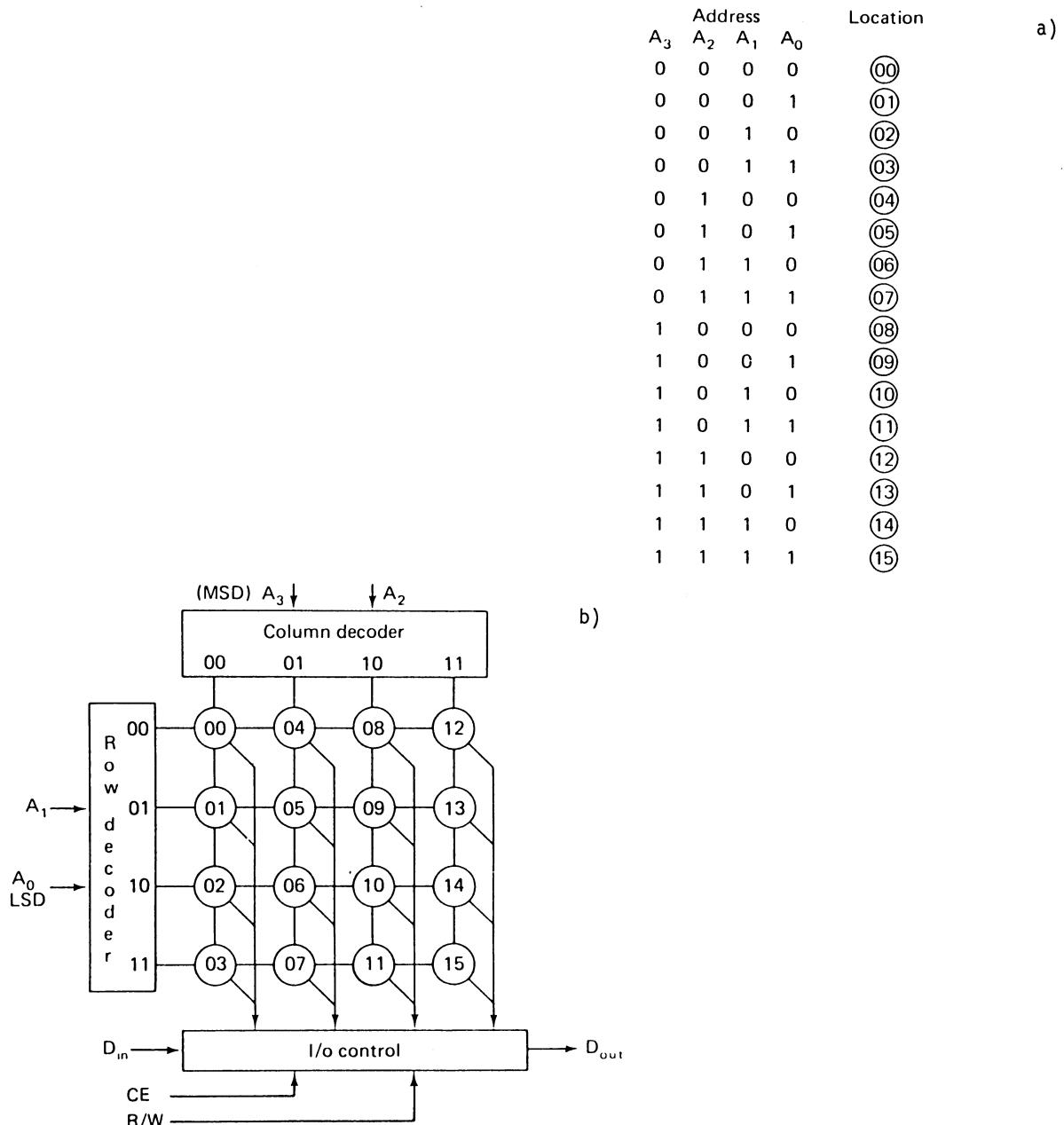


Fig. 23 General ideas about memory. a) Tabular form (linear selection); b) Matrix form (coincident selection).

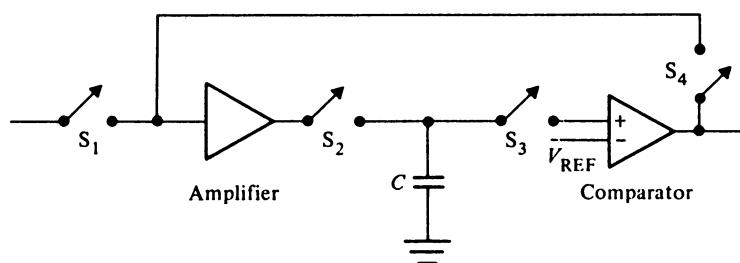


Fig. 24 Symbolic diagram of the structure of dynamic memory

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

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

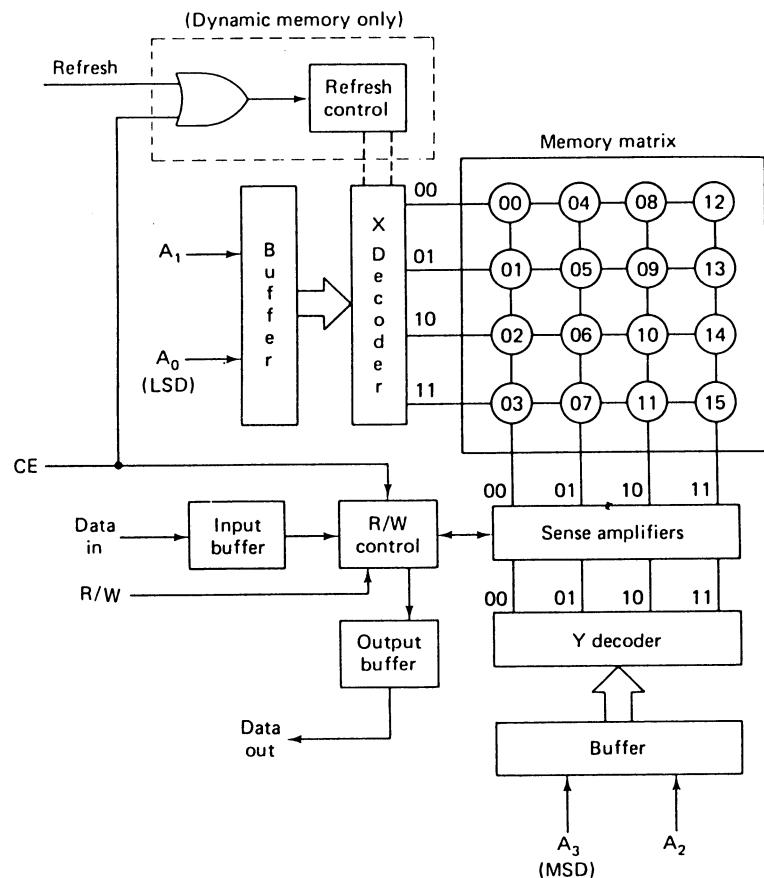


Fig. 25 More detailed picture of the structure of a generic RAM chip

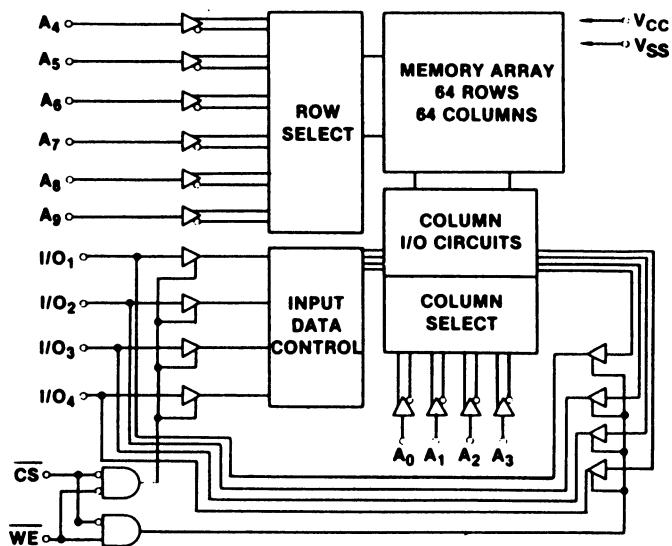


Fig. 26 A 4 K-bit chip. A 64×64 array organized as 1024 addressable words of 4 bits.

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

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

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

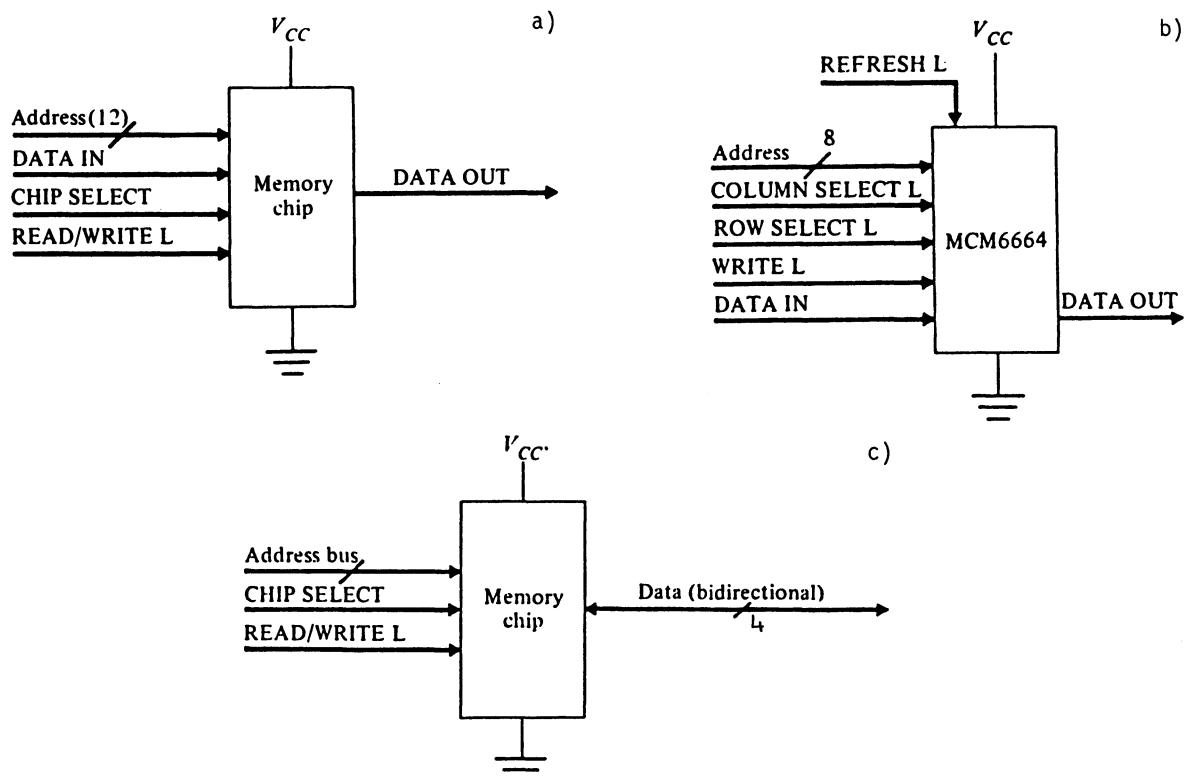


Fig. 27 a) $4 \text{ K} \times 1$ bit static RAM chip; b) $64 \text{ K} \times 1$ bit dynamic RAM; c) 4 K-bit static RAM arranged as 1024×4 bits.

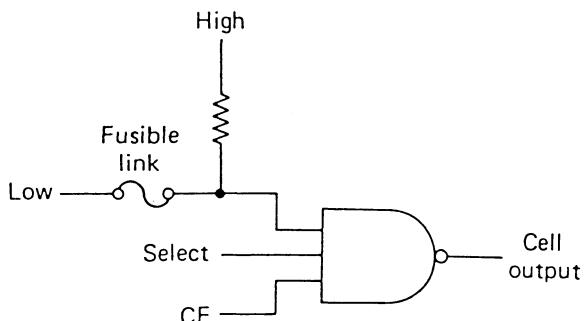


Fig. 28 Programmable Read Only storage cell

altered several times. Another important step was taken when the electrically erasable programmable ROM was introduced. Instead of removing the chip from its socket and exposing it to UV light the PROM could be selectively reprogrammed in situ. The E²PROM therefore acts as a Read/mostly Write/sometimes storage element. The programming, or Write time, is typically ~ 10 ms. The storage will be retained for about 10 years but only a total of 10^4 Erase/Write cycles can be performed. The term ROM is often used as a generic term for all types of read only memory.

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

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

7. PARALLEL INPUT/OUTPUT

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

- a) A parallel data path one or several bytes wide.
- b) An input latch which keeps data valid long enough for the microprocessor to read them.
- c) An output latch to freeze the output data for long enough so that the external device can use them. Remember that a processor has stable data on its bus only very fleetingly.
- d) Control and status bits which inform the external device and processor about the availability and flow of data being transferred via the input and output latches. For example, the processor needs to tell a device there are valid data in the output latch; the device needs to inform the processor when it has taken the data and is ready for a new transfer.

- e) Various occurrences of control and status bits may need to cause interruption of the microprocessor; for example, the availability of a new byte of data from an external device may be signalled via an interrupt.

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

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

- a) The chip contains several independent byte-wide data paths called ports. Two or three are usual.
- b) Data lines are programmable as to direction. Each line or group of lines can be defined and used as an input or output line by loading appropriate control registers within the chip; for example, a direction register containing a Read/Write bit for every data line.
- c) The control and status lines associated with a port may be programmed in an equally flexible way. Various different schemes for handshaking data transfers between a port and an external device may be chosen and implemented. Interruption of the processor may be triggered when a variety of conditions are detected.

In Fig. 29 the upper external connections are to I/O devices; the lower connections are to the microprocessor bus.

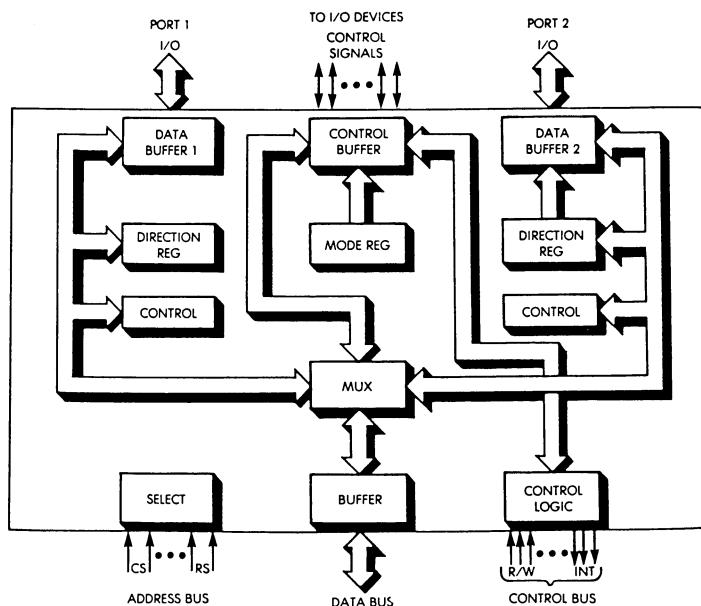


Fig. 29 A generic PIO chip

Complex as they may seem, PIO chips are conceptually very simple. They are useful in many different applications because of their flexibility, and one sees them as the basis for interfacing a wide variety of devices in microprocessor systems.

8. SERIAL INPUT/OUTPUT

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

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

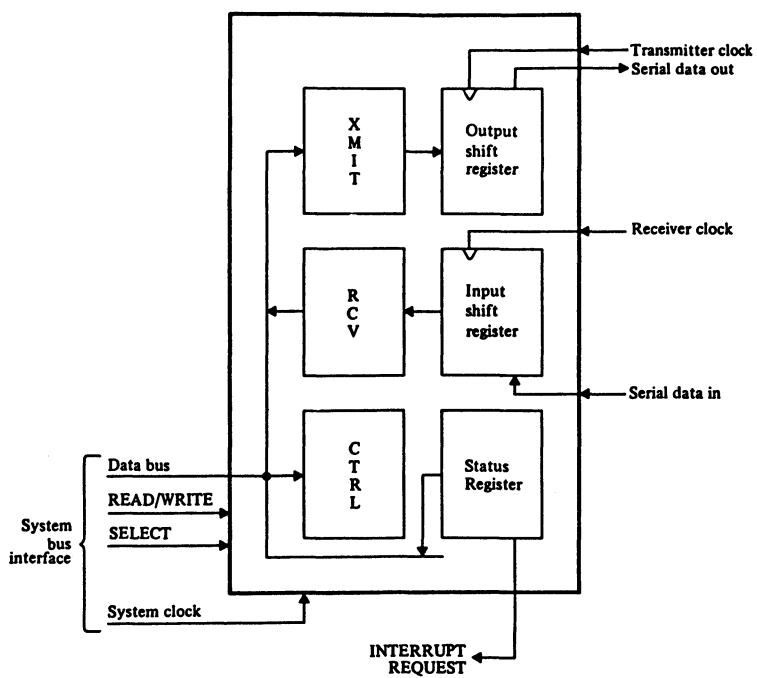


Fig. 30 Typical serial Input/Output Port

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

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

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

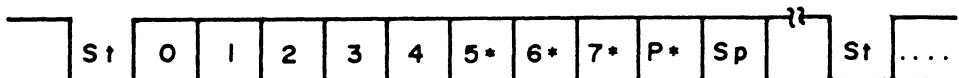


Fig. 31 Timing for byte asynchronous serial transmission: St = start bit; Sp = step bit; P = parity.

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

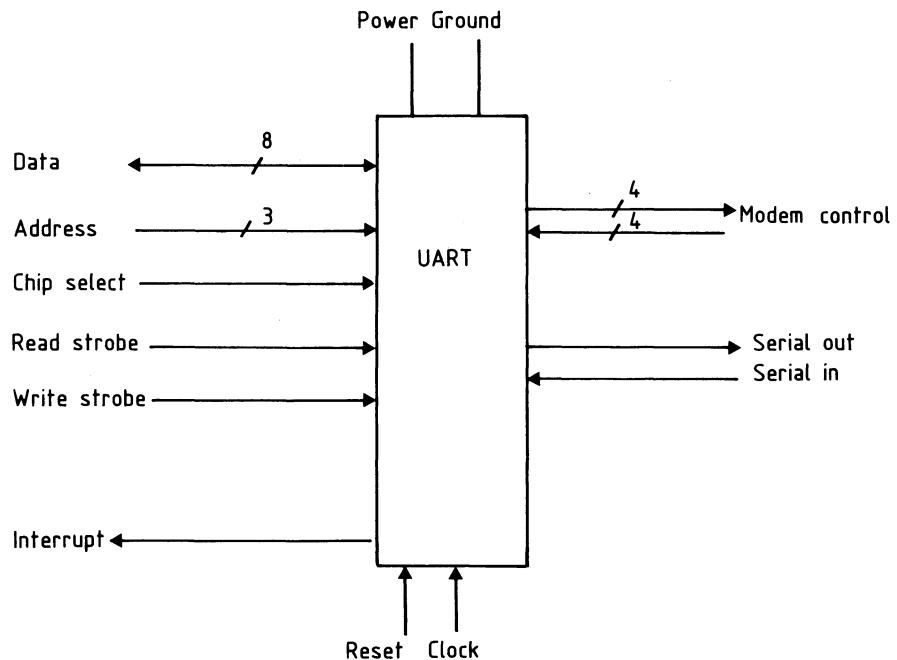


Fig. 32 Simplified diagram of a generic UART

9. LOCAL AREA NETWORKS

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

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

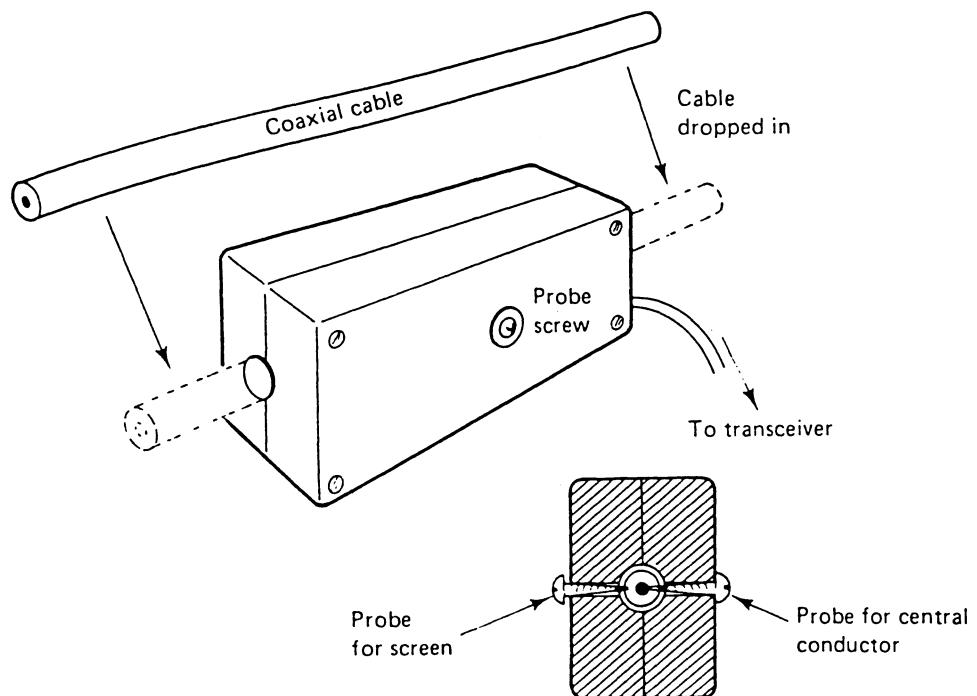


Fig. 33 An Ethernet transceiver

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

Figure 34 shows the format of an Ethernet frame. Let us first discuss the role of the preamble. You will recall in our previous discussion about serial asynchronous transmission that one had to take care to synchronize the local receiver and the transmitter clocks. The start bit not only specified where a byte started but, more important, helped to make sure that the received bit-stream was sampled in the middle of each bit period. There was also a parity bit to aid error detection and a stop bit to specify where the 'byte message' ended. Sending an Ethernet frame necessitates consideration of similar points even though we are now sending bytes synchronously at a well-defined rate instead of one at a time (asynchronously). One difference we meet immediately is the way in which the clocking of data is arranged at the receiver end of the transfer of a frame between two stations. Instead of there being separate receive and

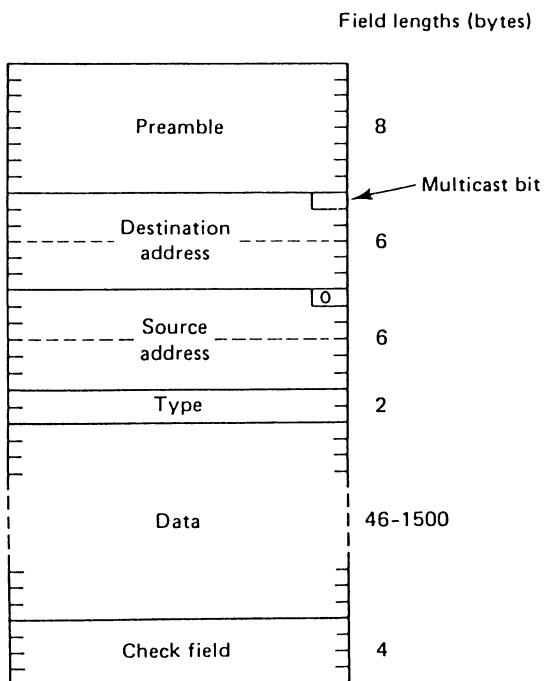


Fig. 34 Format of an Ethernet frame

transmit clocks, the transmitter sends both the data and a clock. This is done on a single information channel, the Ethernet cable, using a technique called Manchester encoding, shown in Fig. 35. The preamble consists of a total of 8 bytes, 62 bits of alternate '1's and '0's which allow clock synchronisation to be established followed by 2 '1' bits which mark the end of the preamble and the beginning of the frame proper. The next fields are the destination and source addresses; both of these are 48-bit quantities. A station recognizing a destination address as its own will accept the frame currently being broadcast on the cable. If the address does not match, then the frame will be discarded.

In addition to specific station addressing, frames may be sent in a multicast addressing mode that is targeted to more than one destination. The source address is

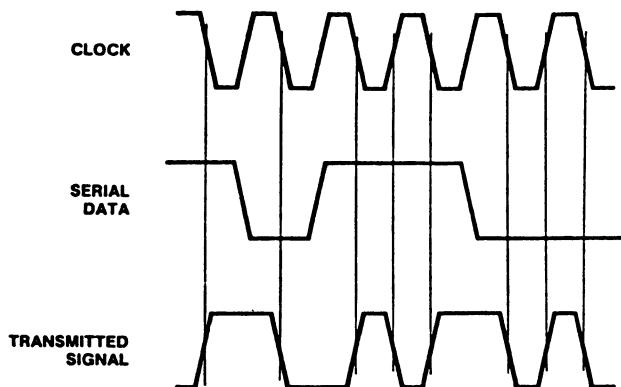


Fig. 35 Manchester encoding of clock and data

necessary so that a receiver knows which station on the network the frame has come from. Note that a station can send packets to itself -- this is a very useful diagnostic tool. The type field is a 2-byte field which can be used to qualify the meaning of the frame in some way. For example, it could give the number of bytes present in the data field or identify one of several different types of frame. The data field, which can be between 46 and 1500 bytes long, is followed by a 4-byte check sum. This is a 32-bit cyclic redundancy check which is computed and appended by the source, over all fields except the preamble, and checked by the destination to determine if any errors have been introduced during transmission.

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

Type A functions

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

Type B functions

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

Type C functions

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

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

The complexity of connecting to Ethernet has led to the manufacture of special Ethernet chip sets which implement, in silicon, all the functions A, B, and C. At the

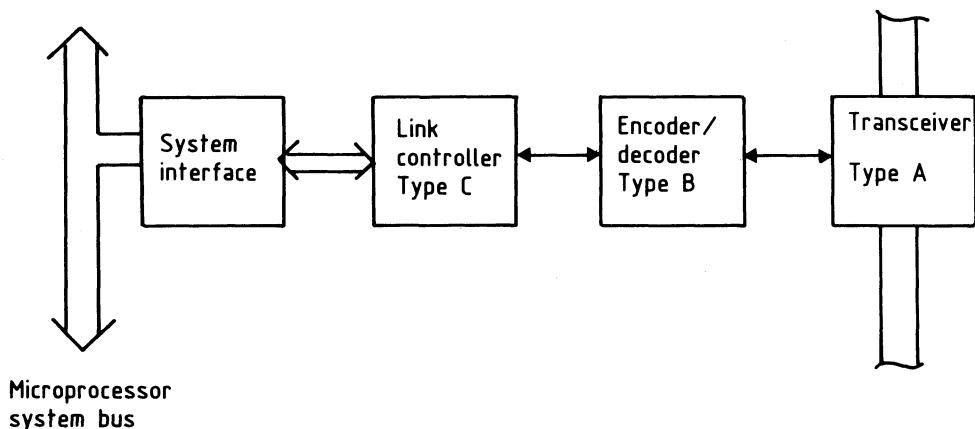


Fig. 36 Interfacing to Ethernet

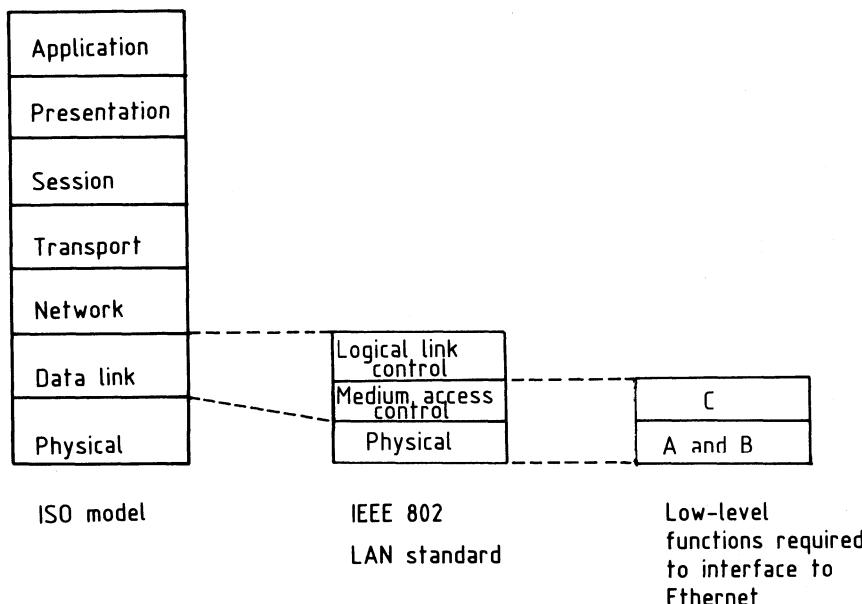


Fig. 37 The ISO model, IEEE 802 and low-level functions required to interface to Ethernet. Many of the functions A, B, and C are being implemented in silicon.

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

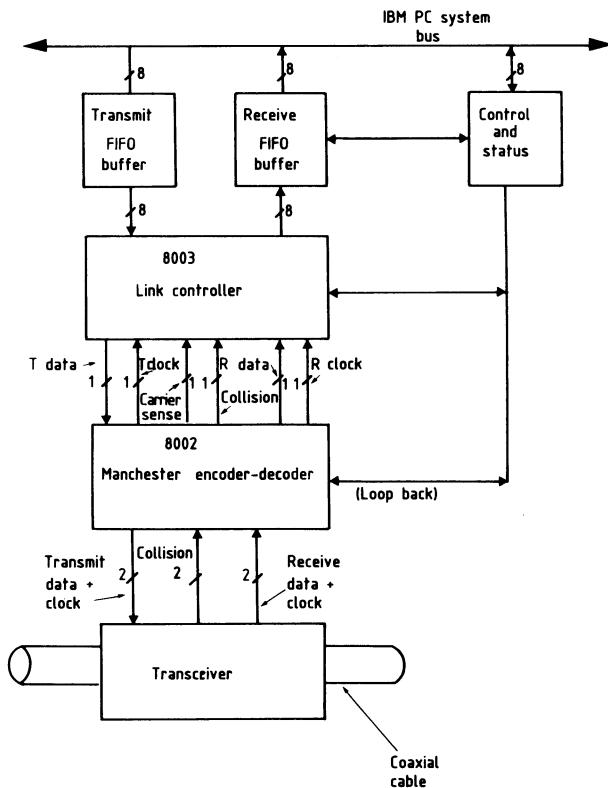


Fig. 38 Ethernet interface (SEEQ chips)

10. VIDEO DISPLAYS

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

The CRT displays we discuss here are called raster displays because a picture or text is produced on a screen by a beam of electrons which scans a uniform pattern of closely spaced horizontal lines. The screen, which is covered with a phosphor, glows when the electron beam hits it. Pictures are formed on the screen as the beam turns off and on and sweeps across its face. Figure 39 shows the basic principles of a CRT and how a raster scan pattern is created. Note how the rising portion of the sawtooth waves applied to the horizontal and vertical deflection system sweep the beam left to right and top to bottom. During the rapidly falling portion the beam retrace to the left and top. For the duration of the retrace the beam intensity is greatly decreased so as to be invisible. The beam is thus said to be blanked on retrace. To avoid flickering, the screen is refreshed at 30 to 60 Hz. An ordinary TV set is the most familiar raster graphics device you will have met.

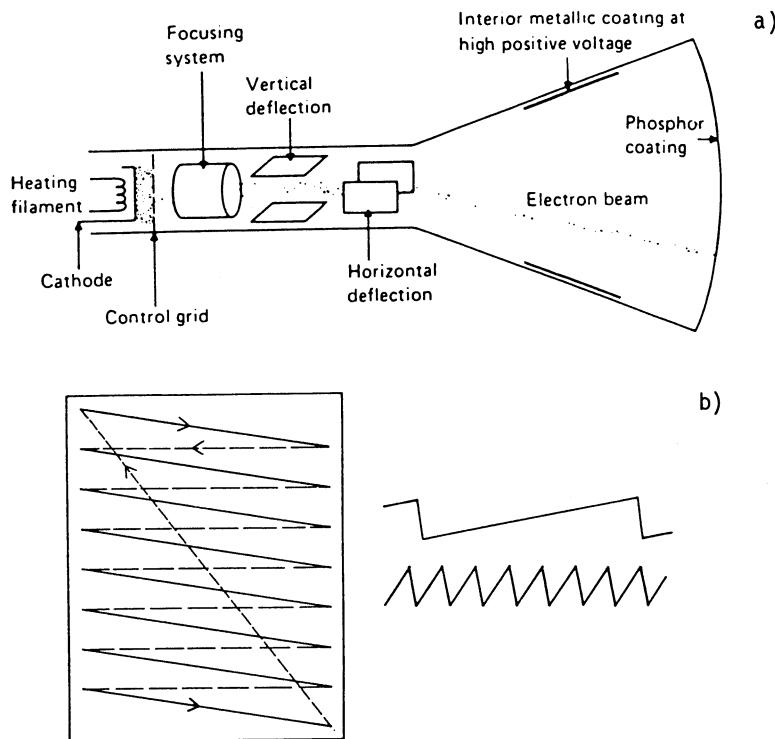


Fig. 39 Basic principles of a CRT. a) The CRT cathode tube itself; b) Details of a raster scan pattern.

Colour displays work in a very similar way except that there are three electron beams, and the screen, instead of being uniformly coated with a single phosphor, is covered with triads of red, green, and blue phosphors. Each of the three electron beams illuminates only phosphor dots associated with one of the three colours. This is arranged by using a shadow mask as shown in Fig. 40. If the red, green, and blue dots are equally illuminated you see a white dot. By varying the relative intensities of the three beams, any colour you want can be produced. Getting an electron beam to strike only the intended phosphor dot and not an adjacent one is a complex and difficult manufacturing task. The

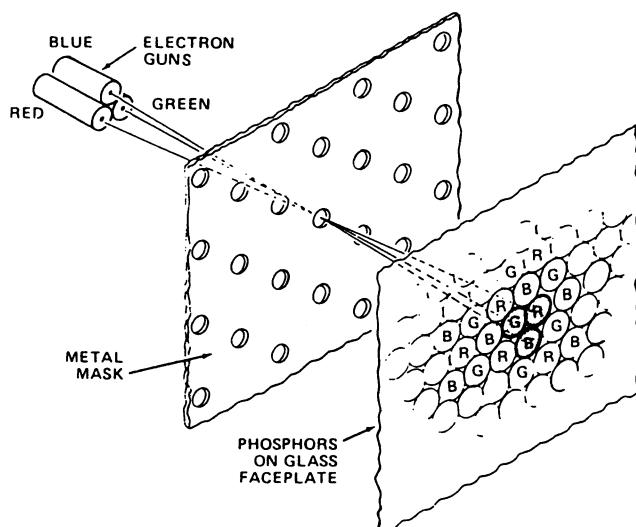


Fig. 40 A colour CRT

highest resolution display around at the moment, which seems to be only a prototype, is 1700 (horizontal) \times 1280 (vertical) dots. A typical low-resolution display has 300 \times 200 dots.

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

The information per scan can be quite large. In the following, let us assume a relatively modest 300 \times 200 pixel array. If each pixel has four bits of information associated with it and the screen is scanned at 60 MHz then the data transfer rate from RAM to the display is 1.8 Mbytes per second. Moving this data with a microprocessor is a regular, heavy, and in some cases impossible load, even if a DMA chip is used (see later). The usual arrangement these days (see Fig. 43) is to use a special-purpose CRT

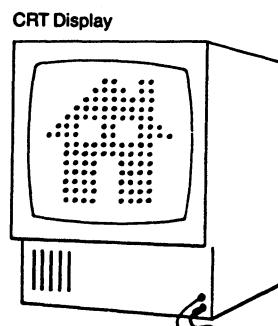


Fig. 41 CRT screen divided up into picture elements or pixels

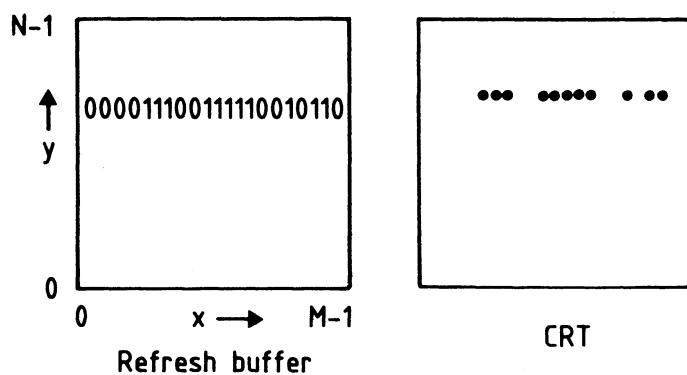


Fig. 42 Use of bit map contained in refresh buffer to modulate electron beam scanning CRT

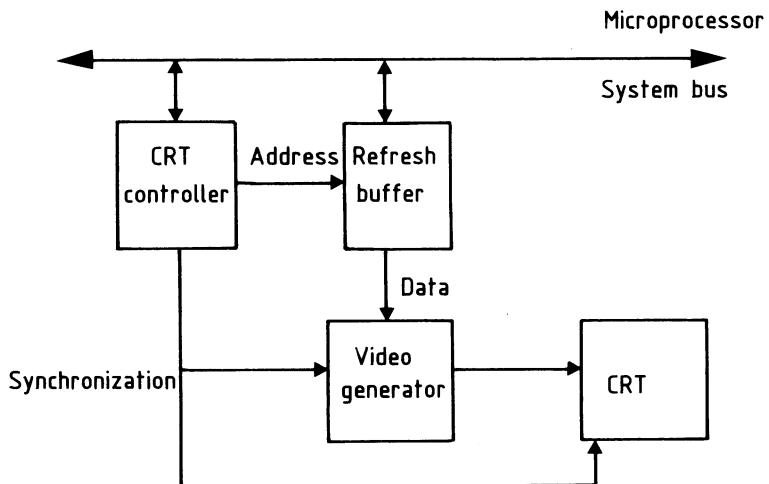


Fig. 43 Much-simplified diagram of how a CRT controller is used to generate a raster scan picture on a CRT

controller chip which shares access to the refresh RAM with the microprocessor system bus. The controller continuously generates the correct sequence of addresses for accessing pixel information, and in addition produces pulses which can be used to synchronize the raster scan for the CRT. Various types of scan can be programmed by loading registers internal to the chip. Some CRT chips also contain facilities for a cursor output and a light-pen input.

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

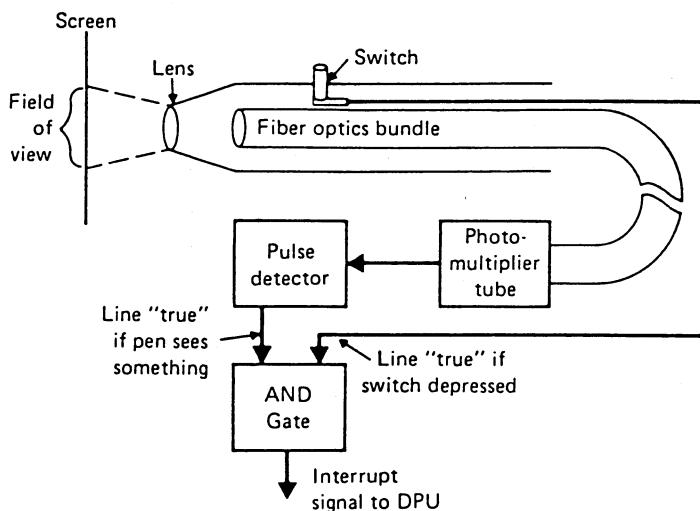


Fig. 44 A light-pen

the refresh memory; an interrupt can also be generated. This value can then be read back from the chip by the host processor.

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

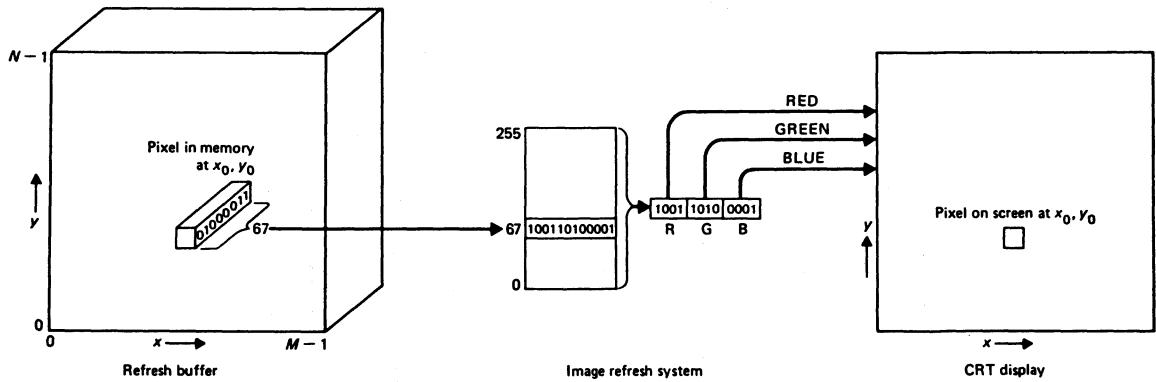


Fig. 45 Use of colour 'palettes'

11. KEYBOARDS

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

Figure 46 shows a very simple keyboard of 16 keys driven from a microprocessor system bus via parallel I/O registers. The keys are arranged as a 4×4 matrix. In the

illustration the black key has been pressed and its closure must be detected. To do this, one uses input and output registers, more normally in practice parts of registers, to implement a keyboard scan. This is done by a key scanning program activating one column at a time with the output register, and reading the resultant row lines via the input register. When a key is depressed it shorts a column to a row. When the column is activated so is the row. The combination of column output pattern and row input pattern uniquely identifies the key that has been pressed. Figure 47 illustrates how a keyboard scan is performed.

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

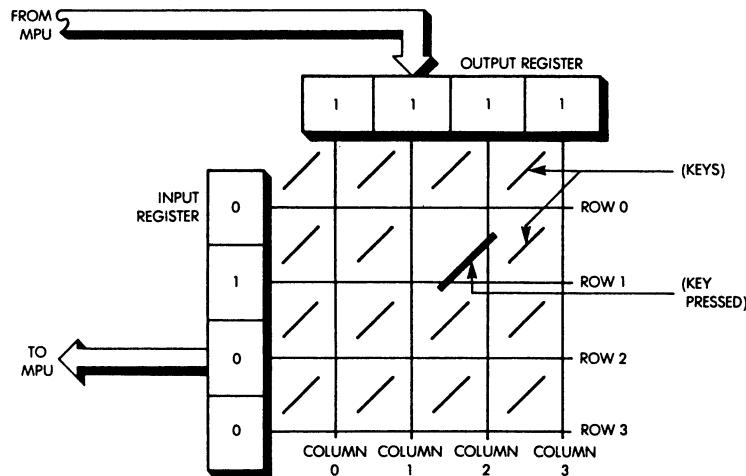


Fig. 46 A very simple keyboard

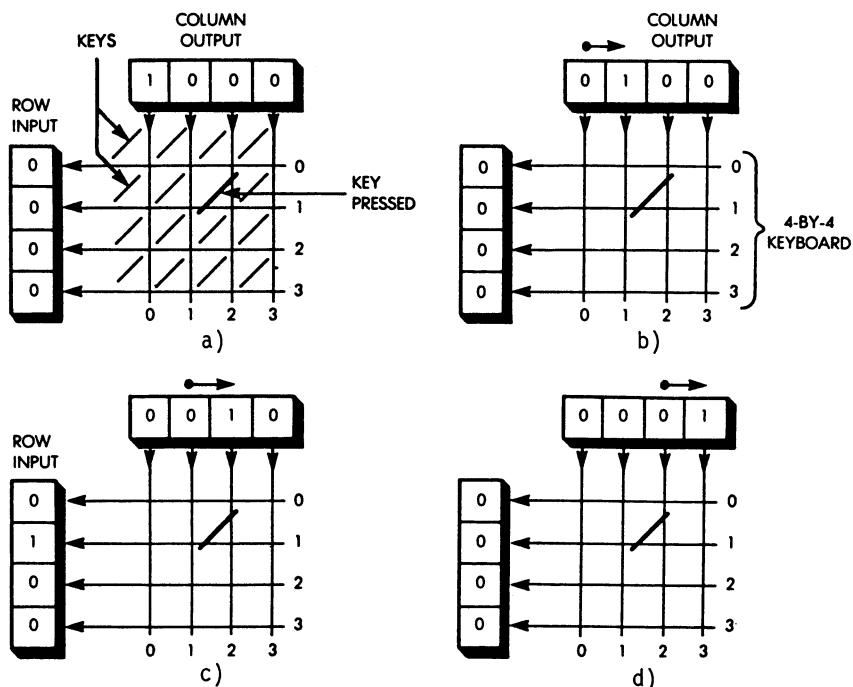


Fig. 47 How a keyboard scan is produced

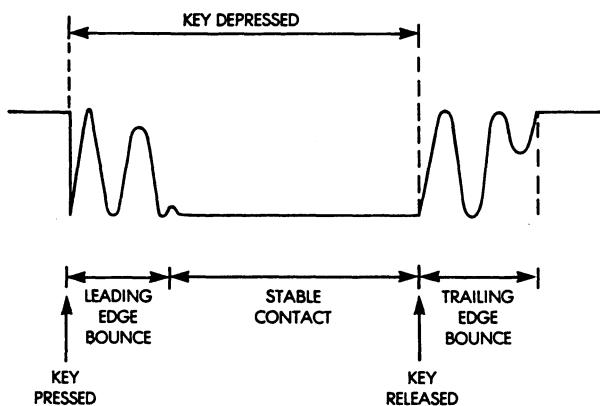


Fig. 48 Contact bounce

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

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

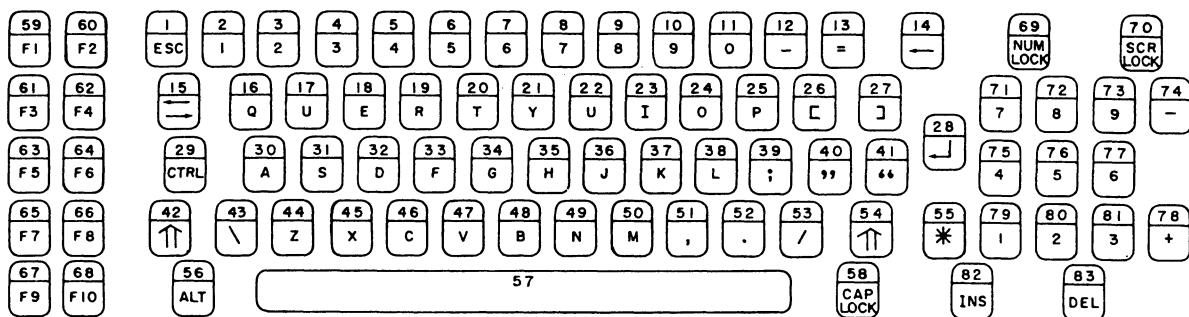


Fig. 49 Keyboard from the IBM personal computer

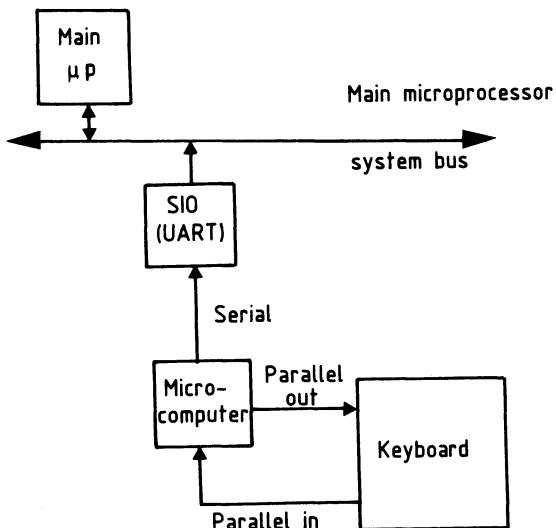


Fig. 50 Keyboard scan using dedicated microcomputer chip

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

One final comment should be made. You will see that I have treated the topics of CRT display and keyboard separately. Many of you will note immediately that the VDU terminals you use on mainframe and minicomputers are a display and a keyboard packaged in a single box. You may wonder what is the relationship of your VDU to what I have said about bit-map displays and keyboards. Figure 51 shows that what is inside your VDU is in fact just these two components. However, there is a difference: the coupling between the

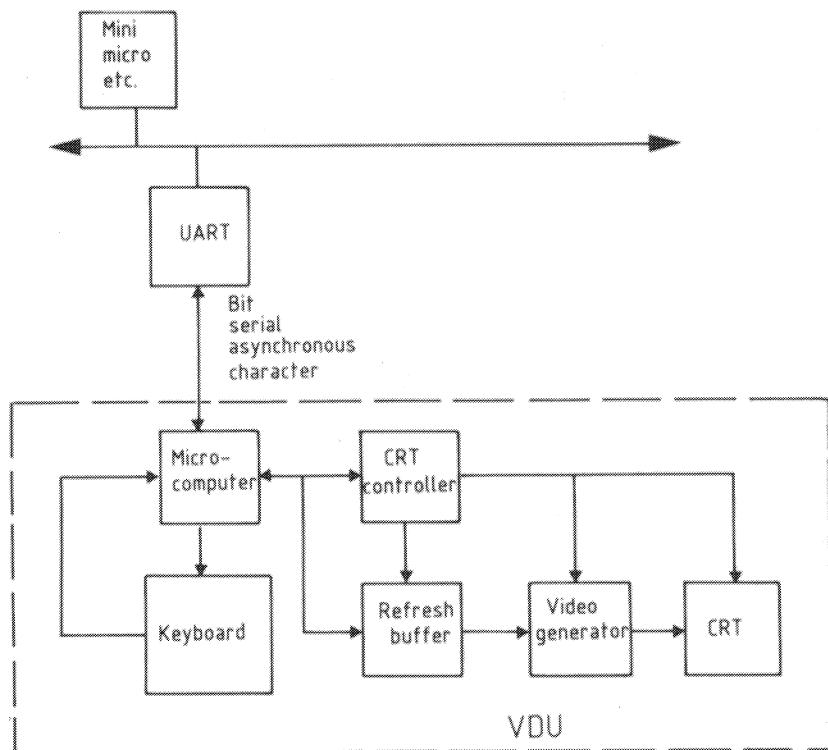


Fig. 51 Simplified diagram of microcomputer based VDU

refresh memory and main processor is weak. By weak I mean that it goes via a slow-speed serial link and is not directly accessible via the host's system bus. This introduces several limitations on graphics-type applications. You can change the refresh memory at the rate of only 1 bit per $100 + 200 \mu\text{s}$ instead of 16 bits every few microseconds. Hence animation is just not on -- you can not play PACMAN on your VDU. It is only set up to display characters, and in any case the rate at which you could change the picture would not make for a very exciting game.

12. OF MICE AND MEN

Like many of the ideas behind the present generation of personal computers and professional work stations, the digital 'mouse' has its origins in Silicon Valley back in the 60's. The idea behind the mouse is to provide a convenient mechanism for a user to point to different parts of a CRT screen and to signal to the computer responsible for running the display that some action is necessary. The philosophy behind the mouse and other 'point and pick' devices such as light pens is to provide a more friendly interactive interface between computers and their users. In systems using a mouse-type of device the role of a conventional keyboard is de-emphasized. For example, instead of having to type a series of commands to initiate a program, a user would be given a menu of possible choices containing not just alphanumeric characters but also small pictures called icons. Icons represent some concept or object; they can tell you what they represent better than words alone. A user would then select an option by simply pointing to it. Figure 52 shows a display generated by an Apple Macintosh which illustrates the

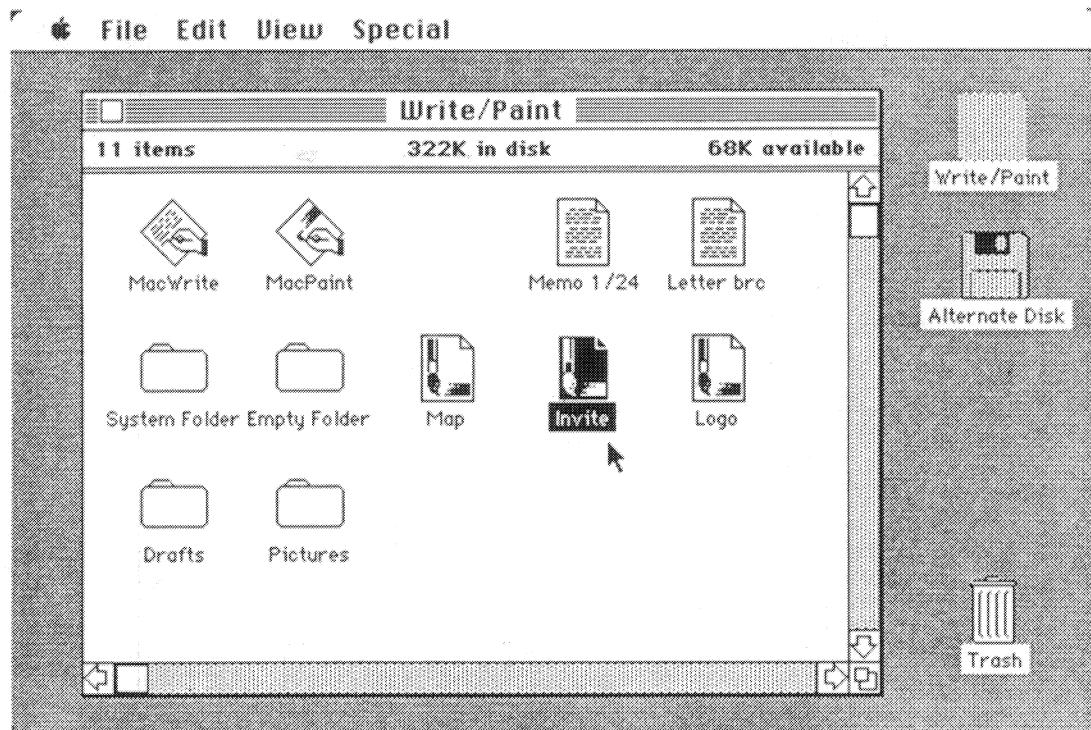


Fig. 52 Menus and icons on the Apple MacIntosh

concepts of menus, icons, and choosing through pointing. As I do not want to steal material from Ian Williams I will say no more about the display side of things, but instead go on to explain how a simple form of mouse is constructed and interfaced.

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

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

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

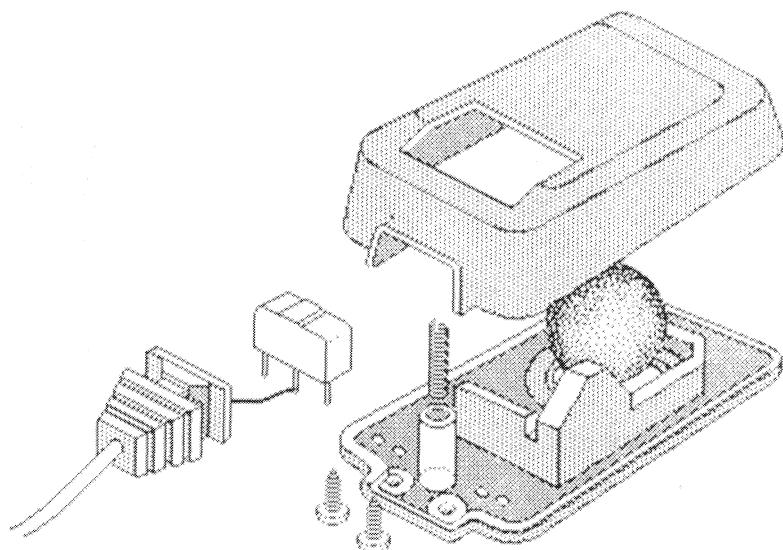


Fig. 53 A raster graphics CRT display of a mouse

13. PRINTERS

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

Most dot matrix printers, like letter quality printers, use an impact method for printing. The dot matrix pins hit an inked ribbon which in turn impacts the paper. This scheme is shown in Fig. 55. The main advantage of this type of printer is its speed and versatility (graphics and even colour) compared with daisy wheel models.

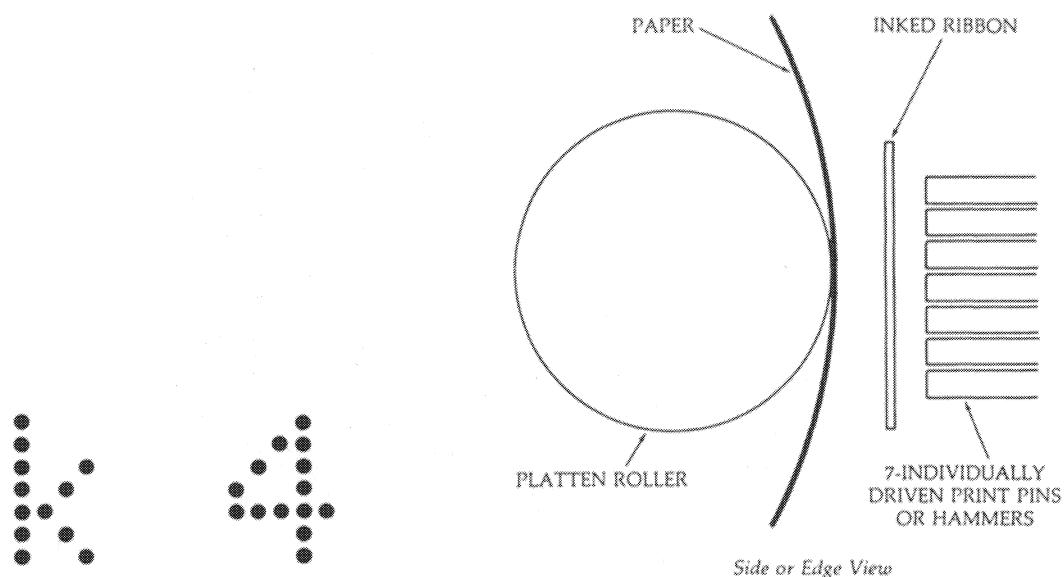


Fig. 54 Dot matrix character formation

Fig. 55 Dot matrix printing

Interfacing a printer is relatively easy using a standard parallel or serial I/O chip. There is even a standard connection between printers and computer systems in the form of the Centronics Parallel Interface. This interface can be simply built from a handful of TTL glue chips. The Hewlett Packard inspired GPIB (General-Purpose Interface Bus) is another possible connection scheme for printers as well as for a whole range of laboratory instruments.

14. FLOPPY DISKS

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

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

Let us now look in a little more detail at the storage format of one floppy-disk system, namely that used on the IBM Personal Computer. Note that other formats offering a factor of 2 more are readily available, and that IBM on their new PC AT have introduced floppy-disk drives with a factor of 4 more storage capacity on the same size diskette.

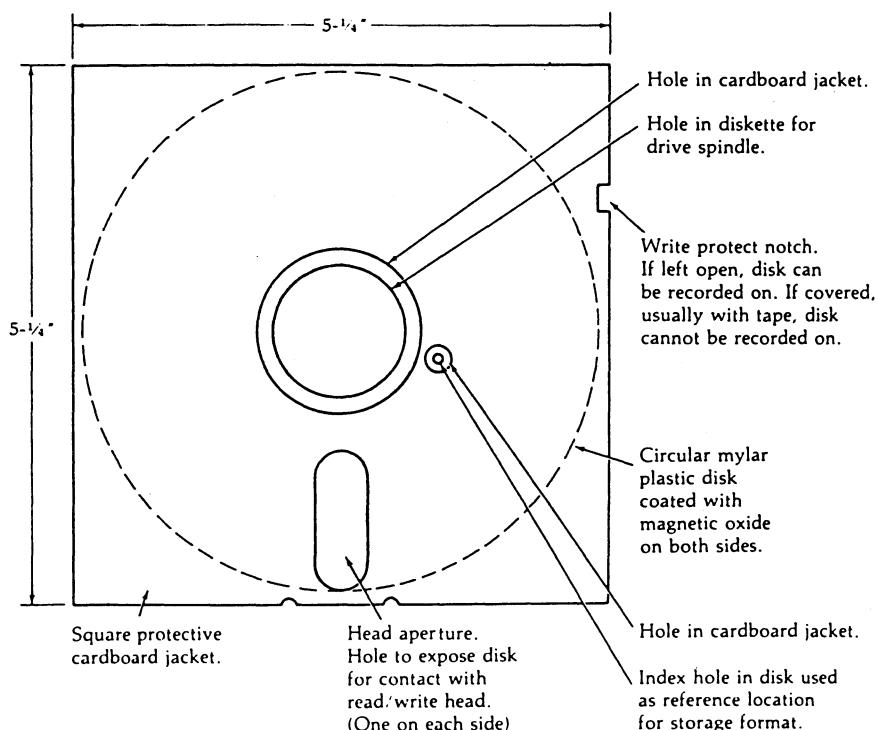


Fig. 56 A diskette

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

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

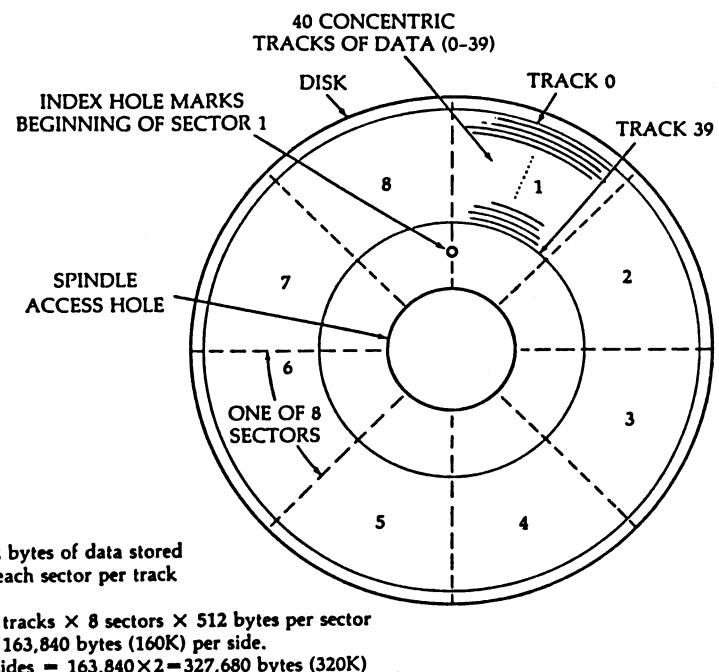


Fig. 57 Format of a diskette

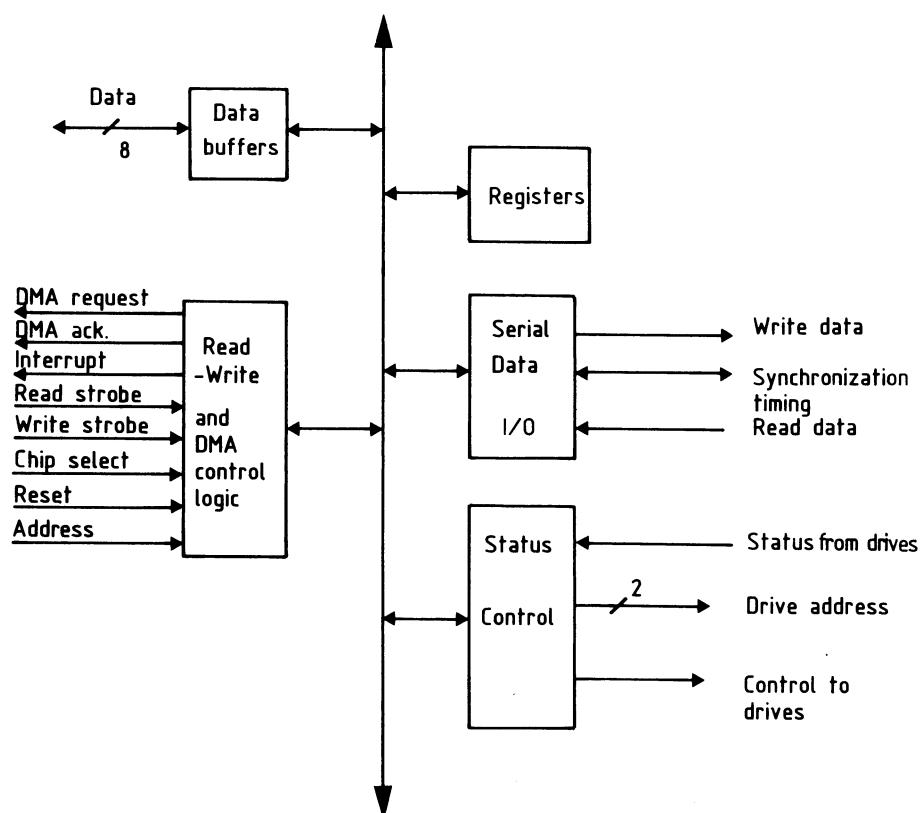


Fig. 58 A disk controller chip

15. DMA CONTROLLERS

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

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

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

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

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

Once the DMA controller has acquired mastership of the microprocessor bus from the CPU, in order to carry out the transfer of a block of data, one asks how long may it retain it. Various possibilities exist. The controller could retain the bus for the duration of the whole block transfer, or at least as long as data were available to be transferred. This clearly maximizes the overall data transfer rate because the overhead in acquiring the bus is amortized over the maximum number of bus transfer cycles. However, this somewhat antisocial behaviour may introduce an unacceptable 'latency of access' for other bus users. A more usual mode is for the controller to request use of

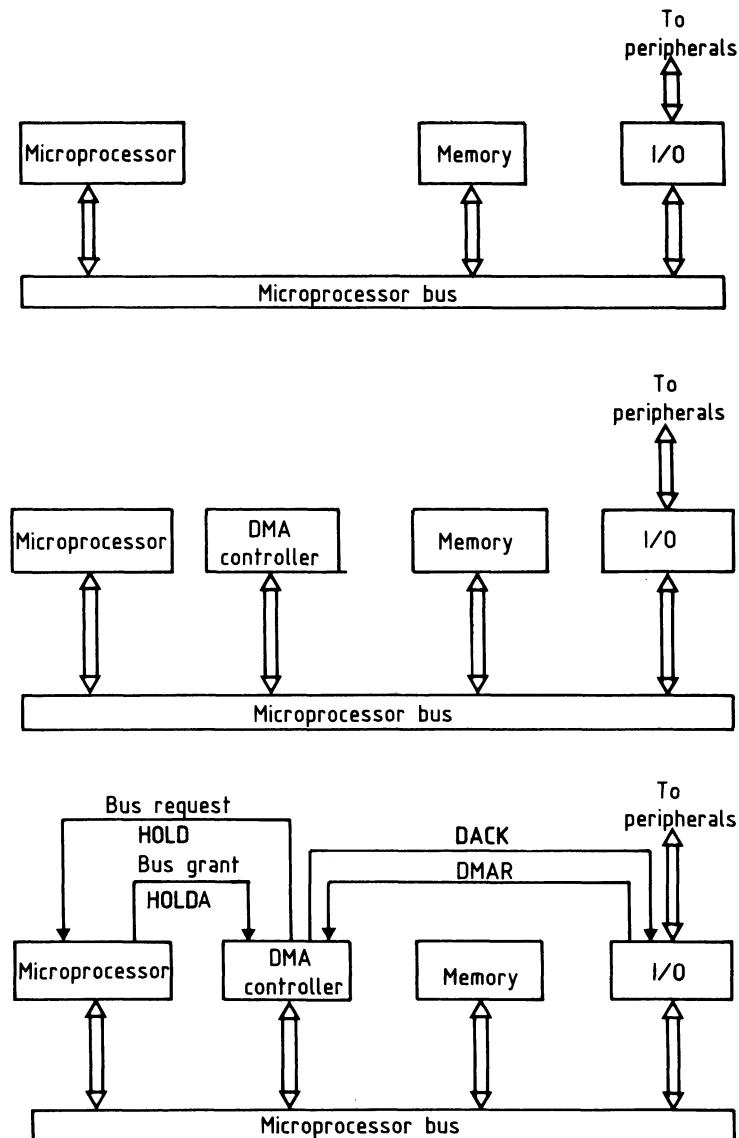


Fig. 59 Adding direct memory access DMA

the bus when data are ready to be transferred and relinquish mastership after one or a few transfers, or when there are no data ready to be transferred -- whichever happens first. We must also consider how data actually flow over the bus during a DMA transfer. One solution is to use a two-cycle DMA and buffer the data in the controller. During one bus cycle the DMA controller accesses the source, storing it in an internal register. In a subsequent cycle the data is written to a destination address. An alternative solution allows a single-cycle DMA transfer between an I/O device and memory. You may wonder how this can be arranged, given that only one address at a time can be present on the address bus. However, the answer lies in the use of the DACK line, which is sent from the DMA controller to the I/O device asserting DMAR when a transfer is taking place. This line is used as an alternative addressing mechanism to select the I/O device. The DMA controller uses the normal address lines to specify the memory location involved in the transfer. Timing is most conveniently arranged by the use of separate I/O and memory strobe signals. One bus cycle can then transfer information directly between an I/O device and

memory. This type of transfer is often called a 'fly by' transfer, as information flies by the DMA controller and is not stored within it. Figure 60 illustrates single- and double-cycle DMA.

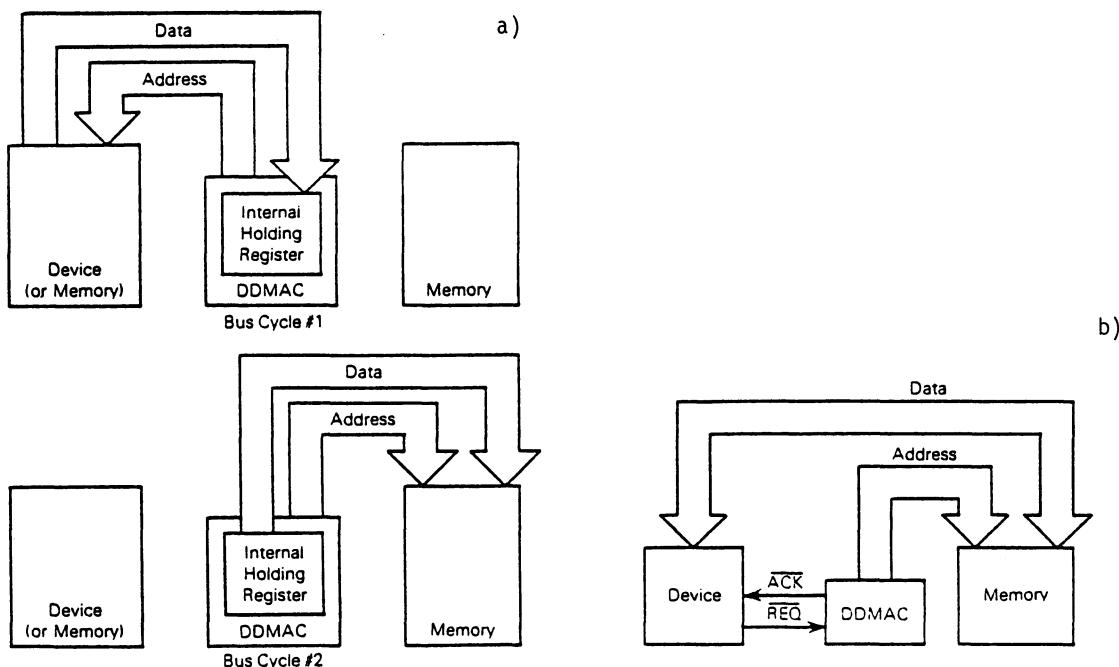


Fig. 60 a) Double and b) Single DMA cycles

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

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

One can identify several other types of device that exhibit DMA controller-like behaviour. One very natural evolution of the standard DMA controllers we have discussed so far is to a so-called smart DMA controller or I/O processor. Along with being able to perform conventional DMA operations, an I/O processor is endowed with the ability to execute its own specialized instruction set. These instructions are fetched from memory and executed in an analogous way to standard microprocessor chips. In addition to merely moving blocks of data around, an I/O processor can be programmed to perform a large variety of other useful jobs: for example, data acquisition, data buffering and

conversion, link protocol management (error checking and re-try), intelligent disk controller, and so on. Figure 61 shows a typical application for an I/O processor. An INTEL 8089 acts as a front end to an INTEL 8086 microprocessor. The 8089 executes its own program under the direction of the 8086. Communication between the two processors is via tables in a shared memory which can be accessed by both parties. The 8089 can relieve the 8086 of much of the I/O load and can intelligently transfer blocks of data at much greater speeds than are possible with the CPU alone. It is interesting to note that the Ethernet chips now becoming available from AMD, INTEL, and others embody within them many of the characteristics of an intelligent I/O processor. We can look for still more intelligence in support devices in the future.

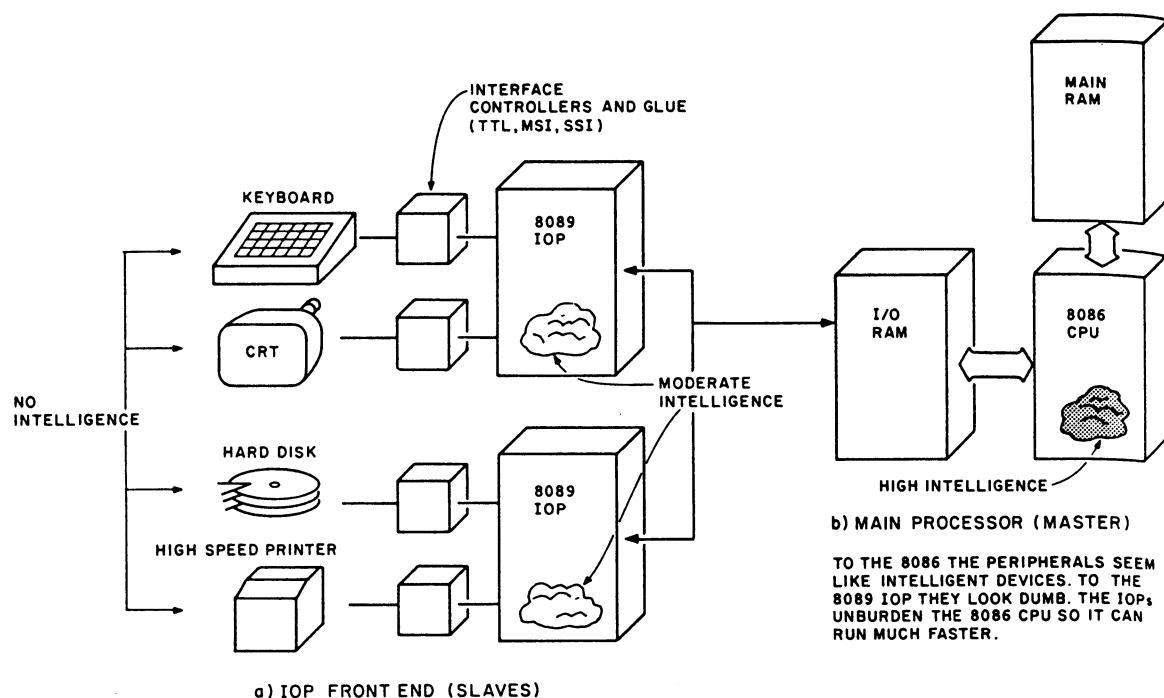


Fig. 61 Use of an I/O processor

16. FLOATING POINT PROCESSORS

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

Figure 62 shows how an INTEL 8087 can be used together with an 8086 microprocessor. As far as a programmer is concerned, he or she is only aware of a single machine.

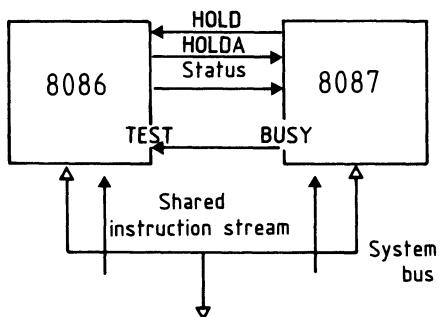


Fig. 62 Use of the 8087 floating-point processor

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

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

INSTRUCTION	APPROXIMATE EXECUTION TIME (MICROSECONDS)	
	8087	8086
ADD SIGN-MAGNITUDE	14	1600
SUBTRACT SIGN-MAGNITUDE	18	1600
MULTIPLY (SINGLE PRECISION)	19	1600
MULTIPLY (DOUBLE PRECISION)	27	2100
DIVIDE	39	3200
COMPARE	9	1300
LOAD (DOUBLE PRECISION)	10	1700
STORE (DOUBLE PRECISION)	21	1200
SQUARE ROOT	36	19600
TANGENT	90	13000
EXPONENTIATION	100	17100

Fig. 63 Performance of the 8087 floating-point processor

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

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

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

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

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

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

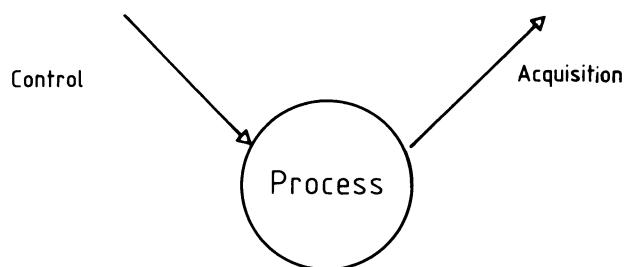


Fig. 64 Simplified representation of data acquisition and control

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

Figure 65 shows the principle of one type of digital-to-analog converter (DAC). It is based on a binary weighted resistance ladder. Each element of the ladder is either connected to ground or to a reference voltage V_{ref} . The state of this connection is conditioned by a digitally controlled switch. If a one is present the switch converts the resistor to V_{ref} . If a zero is present the connection is to ground. For a four-element ladder controlled by a 4-bit number $b_3 b_2 b_1 b_0$, where the b 's are zero or one and b_3 is the most significant bit, the output voltage is

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

hence, V_0 is proportional to the value of the number whose binary representation is $b_3 b_2 b_1 b_0$.

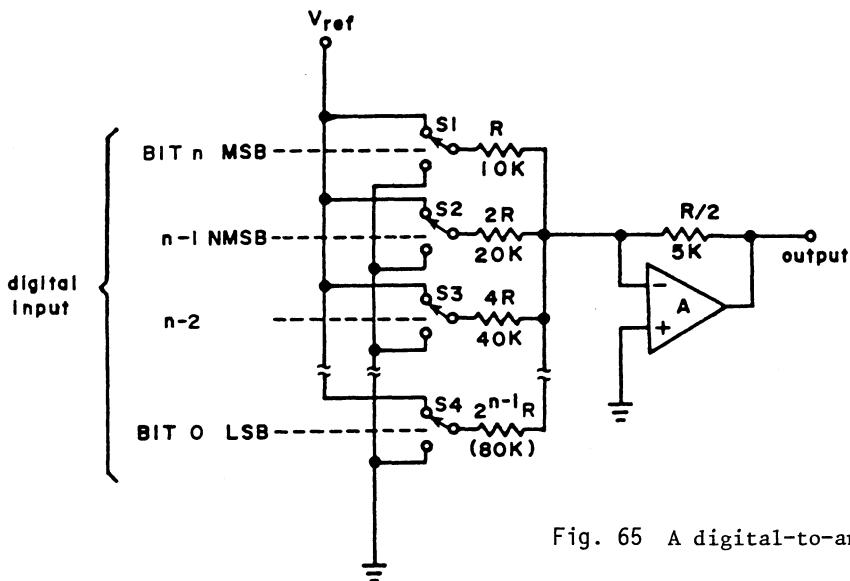


Fig. 65 A digital-to-analog converter

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

There are several methods of implementing analog-to-digital conversion. These methods span conversion times from tens of milliseconds to some tens of nanoseconds for

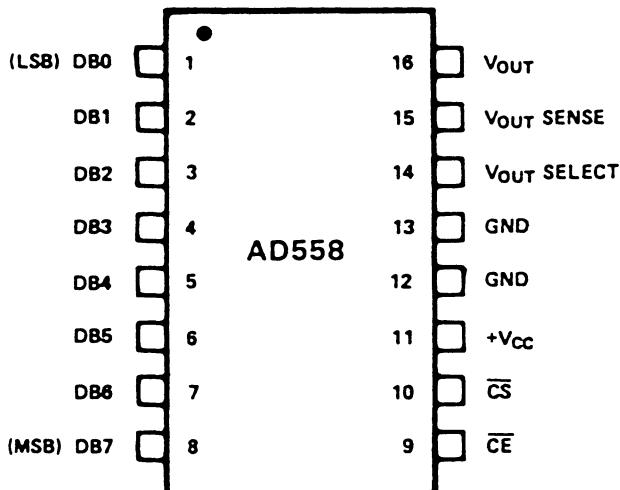


Fig. 66 Low-cost DAC

the ultra fast flash ADCs. We define conversion time as the time to perform the digitalization. All converters except the flash ADC are capable of 12-bit resolution or better. The highest-resolution ADC I could find in the Analog Devices databook had a 16-bit resolution and a 50 μ s conversion time. Flash ADCs have typically a maximum of 8 bits and a conversion time of 30 to 50 ns. We shall look at two types of ADC conversion techniques followed by one well-known chip.

Figure 67 shows a counter or staircase ramp ADC. At the start of conversion the counter starts counting up from zero. The DAC converts the count into a voltage which is compared to the input voltage. When the two are the same the counter stops counting, and its value, which represents V_{in} , can be read. The conversion time can be quite slow, in the 10 ms range. Is there no better way than simply counting up from zero? The answer is Yes, you can use a binary search algorithm. In Fig. 67 we replace the simple counter by a successive approximation register (SAR). At the start of conversion the SAR outputs a value with the most significant bit one and all other bits zero. If the comparator indicates that the DAC output is too big, the bit is removed from the SAR, if not it remains. The next significant bit is then set and the procedure repeated until all bits have been tried out. This type of successive approximation can be both fast, 10 μ s conversion time, and accurate, 12 bits of data.

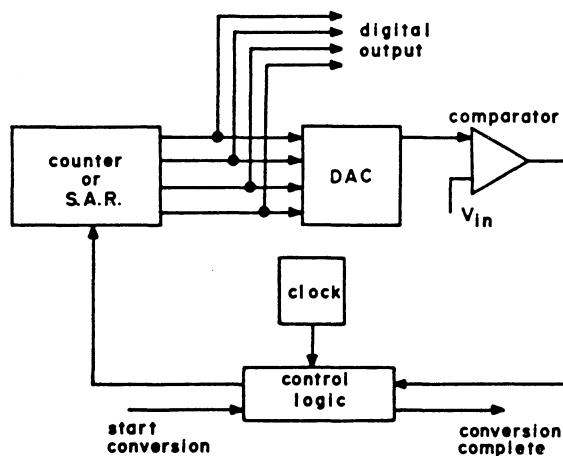


Fig. 67 Principles of staircase ramp/successive approximation ADC

Figure 68 shows the popular National Semiconductor ADC0816 chip. It contains an 8-bit successive approximation ADC with a conversion time of 100 μ s and also a multiplexer which allows 16 input channels to share, in turn, the ADC. The multiplexer channel can be selected using a 4-bit address. Input control signals are provided for latching the input address lines, for enabling the resulting 8-bit digital output, and for starting the conversion process. An end-of-conversion output signal indicates that valid data can be read back from the chip.

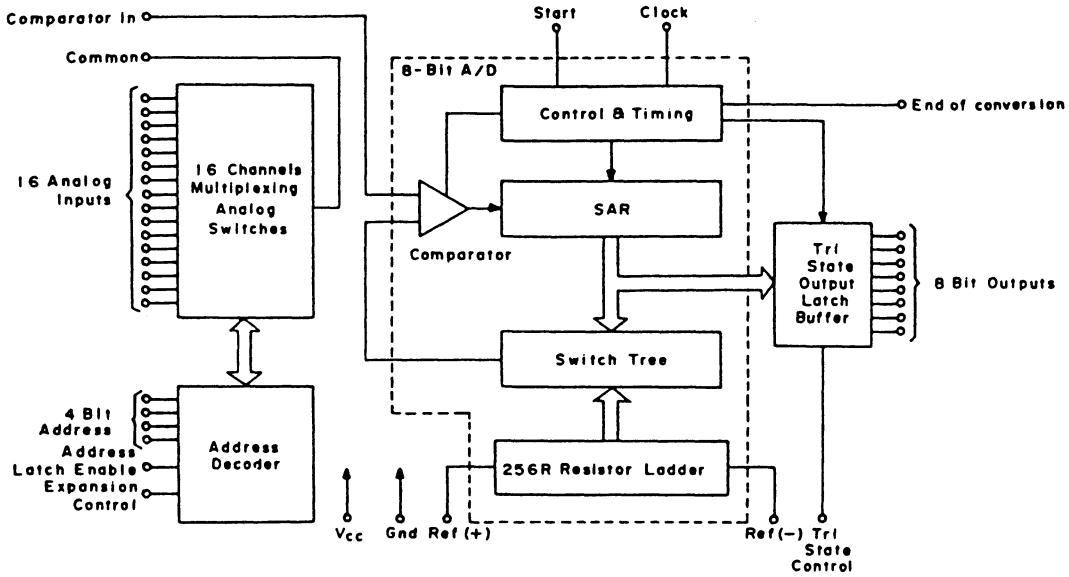


Fig. 68 ADC and 16 channel multiplexer

18. TIMERS

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

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

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

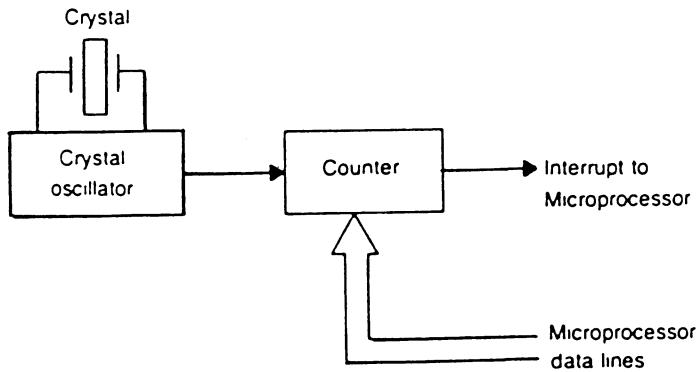


Fig. 69 Simple programmable counter

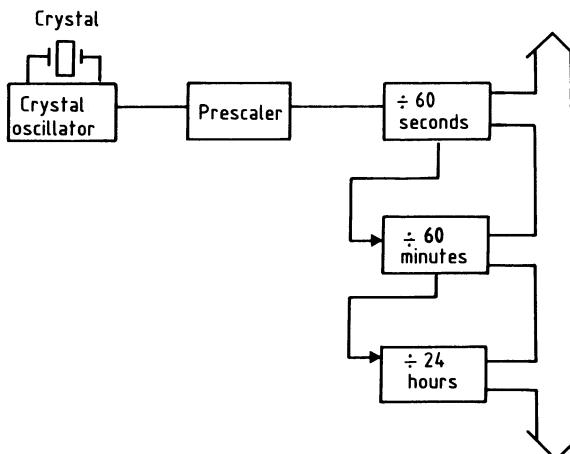


Fig. 70 A simple time-of-day clock

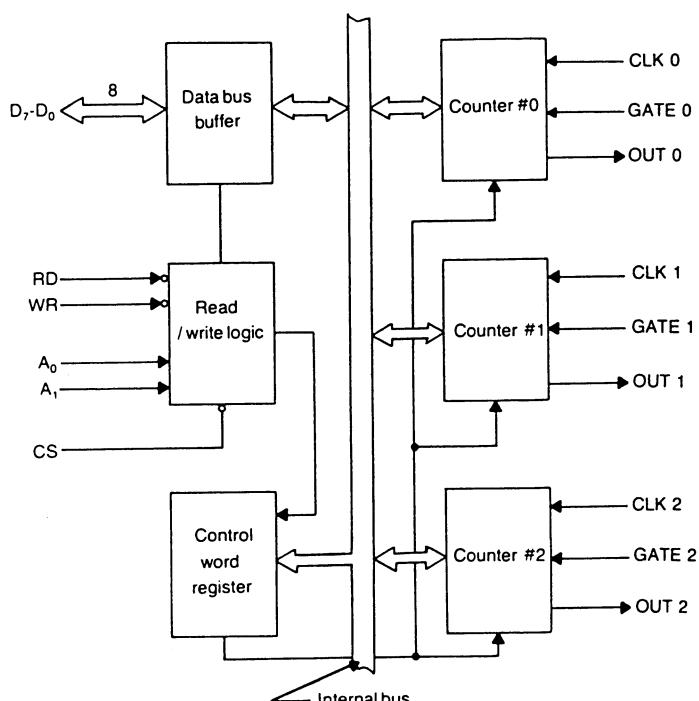


Fig. 71 Programmable interval timer

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

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

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

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

19. PUTTING IT ALL TOGETHER

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

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

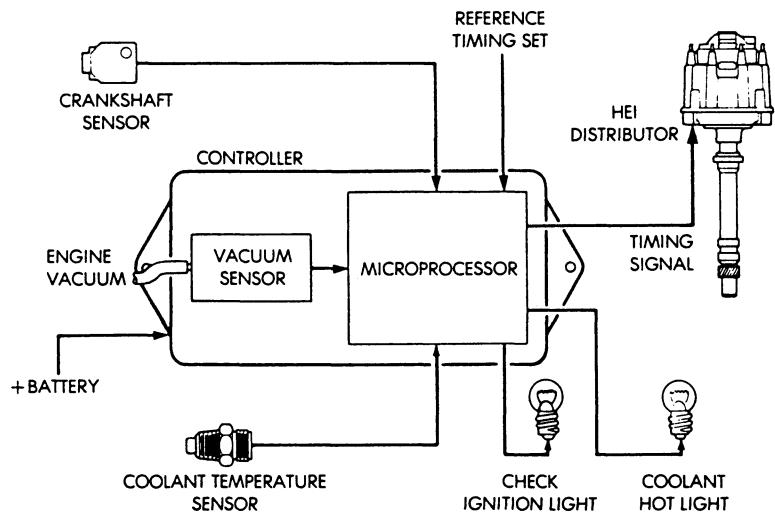


Fig. 72 Microprocessor control of car spark-timing

major advantage of the system is claimed to be the reduction in engine emission (pollution!) because of the precisely controlled nature of the ignition timing.

Figure 73 shows a microwave oven controller. Control is accomplished using a single-chip microcomputer such as the INTEL 8048. There is a 28-key hexadecimal keyboard which is used as the input mechanism to issue basic commands. A user will specify the time the oven is to start cooking, the nature of what is being cooked, and the weight. If unreasonable data are entered e.g. "chicken", "30 lbs.", an audible alarm is set off and the data are rejected. Provided that the data entered are reasonable, the controller will compute and implement the necessary cooking time. Several other functions are provided,

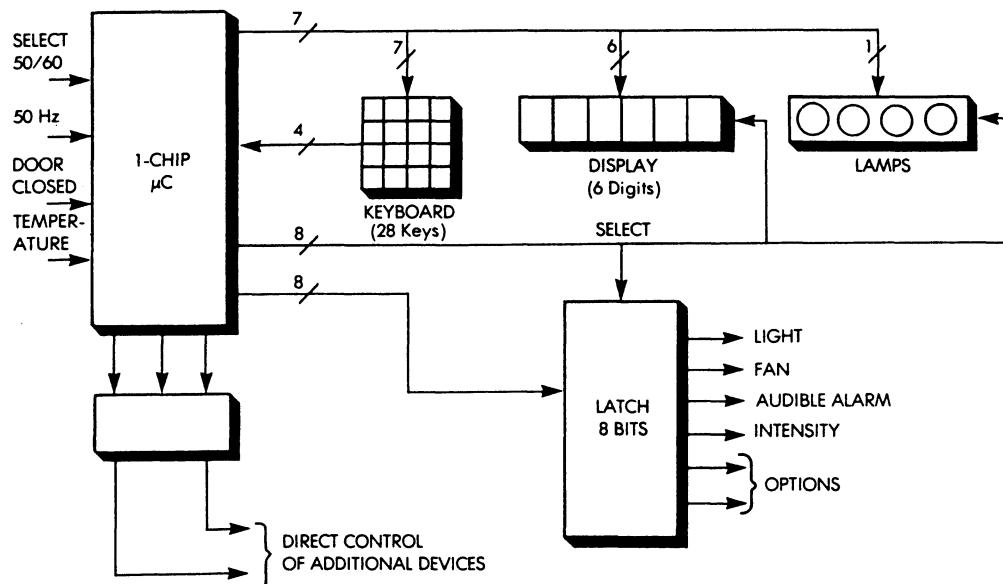


Fig. 73 A microwave oven controller

including a time display, door close and temperature sensing, etc. This area of application is one which has traditionally been carried out in the past by electromechanical methods. Microprocessor control offers the same basic functionality as the older type of control mechanism but with the ability to provide more convenience and features. The 8048 chip, or equivalent, contains -- in addition to an 8-bit CPU -- RAM, ROM, programmable timers, and I/O lines.

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

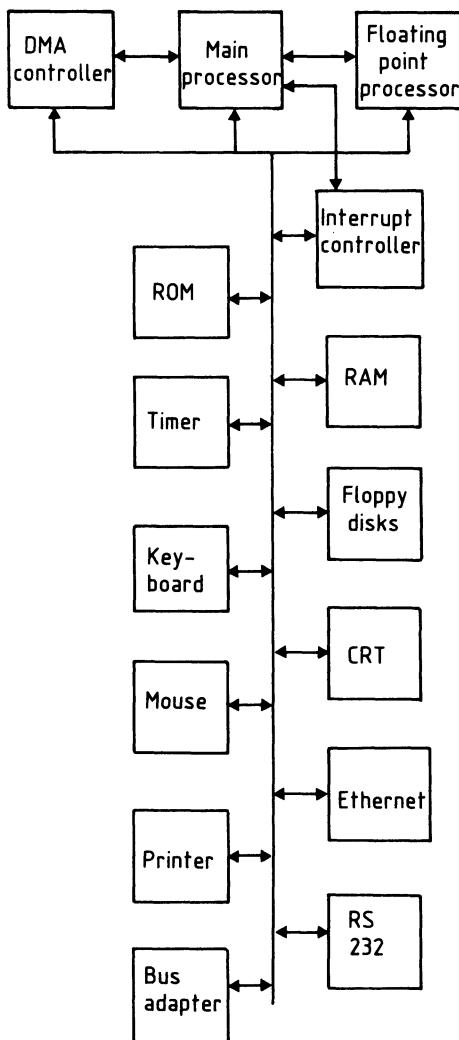


Fig. 74 A generic PC

- The CPU is either a Motorola 68000 family member or one of INTEL's 8086/8, 186/8, 286 series. The processor is capable of addressing around a megabyte of memory. There may even be two standard microprocessors, each capable of running different operating systems and application programs. In one PC the second processor in fact emulates an IBM 370 mainframe.
- There is an optional floating-point co-processor which probably is not too well supported by the available software.

- There is a four-channel DMA controller used for driving the floppy disk and local area network controllers. Perhaps the manufacturer also uses one channel to periodically refresh the dynamic memory.
- There is an interrupt controller allowing eight vectored interrupts to the processor.
- There is 64 kbyte of ROM in which is contained power up diagnostic software, parts of the operating system, and some useful run-time utilities. Almost certainly there will be a version of BASIC running out of ROM too.
- There is a three-channel timer chip which is used by the operating system to keep track of real time and maybe to trigger memory refresh cycles. Users may use the third spare channel as they wish, or the system may perhaps use it for generating audio output (music?) via a loudspeaker.
- The amount of memory on the system will be typically 128 kbytes. If the manufacturer is sensible he will allow for more. The memory will most likely be implemented with 64 k × 1-bit dynamic memory although, 128 k and 250 k chips are arriving. Refresh may be implemented at a system-wide level, for example using a DMA controller channel, or at a local, memory level.
- Secondary storage is typically provided by two floppy disk drives whose capacity may range from 0.3 to 1.2 Mbytes per drive. The drives are operated using a floppy-disk controller chip.
- There is a full-size keyboard and perhaps a mouse to allow the user to interact with his PC. Serial and/or parallel I/O chips are used to interface these devices.
- The CRT screen, which may be colour, is a bit-map raster scan device. Its resolution, which depends on the number of colours available at any time, ranges from 160 × 100 with 16 colours to 640 × 200 for monochrome. The refresh memory can be accessed by the processor as part of its memory space and also by a CRT controller.
- There is an 80 character per second printer interfaced according to the Centronics standard and capable of producing graphical output--a copy of the picture on the CRT for example.
- A connection to a local area network, probably Ethernet or a near relation, has been implemented using one of the newly available protocol chip sets.
- A connection to a wide area network could be made via an RS-232 port and a modem. However, it is more likely that such a function is provided by a separate gateway on the local area network.
- You hope that there is, or will be, a vast reservoir of useful and cheap software of all types, together with glossy publications containing 'useful' articles on how your PC can be programmed in BASIC to play scrabble, to baby sit, to improve your teenage children's grades, and so on.
- If you are lucky or smart in what you buy, then your generic PC comes with clear information as to how to connect equipment to its system bus. You should be able to build or buy boards for a whole variety of applications including data acquisition, plus adapters to standard microprocessor buses and instrumentation buses such as CAMAC and GPIB.

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

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

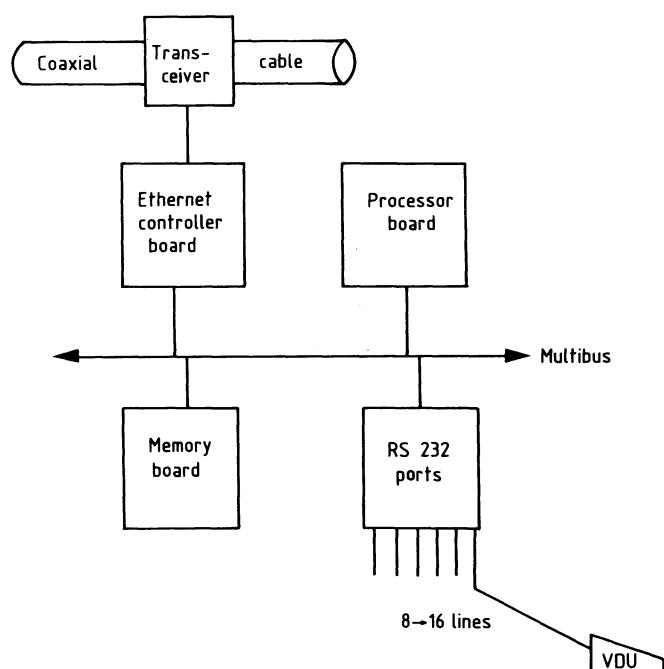


Fig. 75 Synthesizing a terminal concentrator ($8 \rightarrow 16$ lines) for Ethernet using MULTIBUS boards

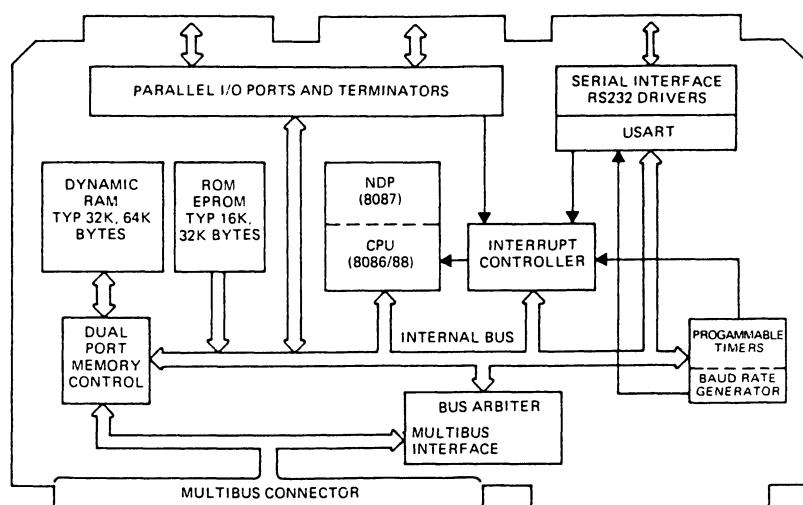


Fig. 76 A MULTIBUS processor board

responsible for traffic in the opposite direction, namely frames sent from different hosts containing information destined for display on multiple VDU screens.

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

ACKNOWLEDGEMENTS

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

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

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

Finally, a sincere thank you to my long suffering wife, Janet, who provided tea, beer, sympathy, etc. at appropriate moments.

REFERENCES

There have been hundreds of textbooks written on the subject of microprocessors; still more magazine articles. The semiconductor industry itself publishes numerous and detailed catalogues of individual components. I have listed below just a few of the books I have read and found useful in preparing these lectures.

J. C. Boyce, Microprocessor and Microcomputer Basics (Prentice-Hall, Inc., New Jersey, 1979).

J. D. Foley and A. van Dam, Fundamentals of Interactive Computer Graphics (Addison-Wesley Publishing Co., 1982).

S. Leibson, The Handbook of Microcomputer Interfacing (TAB BOOKS, Inc., Blue Ridge Summit, Pennsylvania, 1983).

A. Osborne, An Introduction to Microcomputers (McGraw-Hill, Inc., Berkeley, California, 1980), Volume I, Basic Concepts.

M. Sargent III and R. L. Shoemaker, The IBM Personal Computer From the Inside Out (Addison-Wesley Publishing Co., 1984).

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

I. R. Whitworth, 16-bit Microprocessors (Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984).

R. Zaks, Chips to Systems: An Introduction to Microprocessors (Sybex, 1981).

FORMAL ASPECTS IN DATABASES

C. Delobel

Institut de mathématiques appliquées de Grenoble

Abstract : From the beginning of the relational data models special attention has been paid to the theory of relations through the concepts of decomposition and dependency constraints. The initial goal of these works was devoted to the scheme design process. Most of the results are used in this area but serve as a basis for improvements of the model in several directions : incomplete information, universal relations, deductive databases, etc...

CHAPTER 1

DATA MODELS AND PREDICATE CALCULUS

1. Data model approach

From an historical point of view the relational model introduced by Codd (CO70) has appeared as a data model. The intent of this section is to review some basic definitions about the notions of relation and relational scheme. Attributes are symbols taken from a finite set $U = \{A_1, A_2, \dots, A_n\}$. We shall use the letters A, B, \dots , for single attribute, and the letters X, Y, Z, \dots for sets of attributes. Each attribute A has associated with it a domain, denoted by $\text{dom}(A)$, which is the set of possible values for that attribute. The elements of a domain will be represented by lowercase letters, a, b, x, y, \dots . Also, following common notational convenience, we shall let XY denote the union of attribute sets X and Y . A tuple t of type X is an application from t to $\text{dom}(X)$, such that for every $A \in X$, $t(A)$ is an element of $\text{dom}(A)$. A tuple t of type ABC will be usually denoted as a sequence of values, $t = (a, b, c)$. More generally, if t is a tuple of type X and for $Y \subseteq X$ we shall denote by $t[Y]$ the restriction of t to the Y -attributes.

A relation r of type X is a finite set of tuples of type X . A relation is an instance of a relation scheme. By relation scheme we mean both a symbol of name R and a set X of attributes. It is convenient to introduce a function δ which associates for every object its type ; for example, if the tuple t is of type X then $\delta(t) = X$.

A database relational scheme is a pair (R, IC) , where R is a set of relation schemes $R = \{R_1, R_2, \dots, R_n\}$, with $\partial(R_i) = X_i$, for $1 \leq i \leq n$, and IC a set of integrity constraints. Integrity constraints are sentences concerning the relations which specify the valid states of the database. A database $r = \{r_1, r_2, \dots, r_n\}$ is a set of relations where every r_i is an instance of R_i and obeys the sentences in IC . When no confusion is permitted, we shall identify a relational schema R from the type of each relation name, thus $R = \{X_1, X_2, \dots, X_n\}$. $REL(R)$ defines the set of databases of type R , and $SAT(IC)$ the set of databases which satisfies the integrity constraints IC .

In the original presentation of the relational model Codd introduced two different types of languages as data manipulation language : the relational algebra and a first order logical language. There are five basic operations that will be of some interest to us : projection, join, selection, union and difference.

Let r be a relation of type X , the projection of r over an attribute set $Y \subseteq X$ is defined by :

$$r[Y] = \{t[Y] : t \in r\}.$$

Let r and s be two relations of type X and Y , the join of r and s is a relation denoted by $r * s$ and defined by :

$$r * s = \{t : \partial(t) = XY \text{ et } t[X] \in r \text{ et } t[Y] \in s\}.$$

Let r and s be two relations of the same type X , the union $r+s$, and the difference $r-s$ are respectively defined by :

$$r+s = \{t : t \in r \text{ ou } t \in s\},$$

$$r-s = \{t : t \in r \text{ et } t \notin s\}.$$

Finally, the selection operation consists in extracting from a relation r the tuples which obeys a boolean expression E , it is denoted $r : E$.

2. First order relational language and relational structure

Mathematical logic is a power tool to describe both the concept of database and data manipulation languages. A first order language is a pair (A, F) where A is an alphabet of symbols and F a set of well formed formulae built from the alphabet and syntactic rules. Primitive symbols of such a language are :

- variable x, y, \dots
- constants a, b, \dots
- predicates P, Q, \dots
- logical constants, $\&$ (and), \mid (or), \sim (not), \Rightarrow (implication)
- quantifiers, $(\forall x)$ (forall x),
 $(\exists x)$ (there exists x)
- functions f, g, \dots

A term is defined recursively to be a constant, a variable, or if f is a function symbol and t_1, t_2, \dots are terms, then (t_1, t_2, \dots) is a term.

If P is a predicate symbol and t_1, t_2, \dots are terms then $P(t_1, t_2, \dots)$ is an atomic formula.

A well formed formula, or more simply a formula, is defined recursively to be an atomic formula, or if F_1 and F_2 are formulae then $F_1 \& F_2, F_1 \mid F_2, F_1 \Rightarrow F_2, \sim F_1$ are formulae. If x is a free variable in a formula F , then $(x)(F)$ and $(Ex)(F)$ are also formulae. A closed formula is a formula where all the variables are quantified.

Relational languages (Re 81) have special properties due to the database area. First of all, we shall consider a finite set of constants and predicates. Among all predicates one is specially distinguished, it is the equality predicate ($=$) for checking if two objects are identical. Symbol functions are eliminated since they induce bad properties related to infinite set of values. Finally among all predicates we shall assume that there exists a finite set of unary predicates associated with every independant domain of values for a variable ; that is that every variable will be typed by a predicate. The type predicate is defined recursively from primitive type as follows :

- a primitive type is a type predicate
- if β_1 and β_2 are type predicates then $\beta_1 \& \beta_2, \beta_1 \mid \beta_2$ and $\sim \beta_1$ are type predicates.

According to these definitions it is convenient to introduce some abbreviation rules for formula expressions :

- $(x.\beta)(F)$ means "for all x such x is β F is the case" and stands for the correct expression $(x)(\beta(x) \Rightarrow F)$,
- $(Ex.\beta)(F)$ means "there exists x in such F is the case", and stands for the correct expression $(Ex)(\beta(x) \& F)$.

Once the language has been defined, it is necessary to assign to every formula a meaning. It is the role of an interpretation to do so. An interpretation I of a set of formulae consists in the specification of a triple (D, K, E) where :

- D is a finite domain of values, for instance the database values,
- K is an application which associates for each constant symbol in A an element of D ,
- E is an application which assigns for each n -ary predicate symbol a relation of D^n , called the extension of the predicate.

In an interpretation we are concerned with the truth evaluation of a closed formula. The truth value of a closed formula is obtained according to the usual rules of boolean calculus. An interpretation of a set of formulae is a model if all the formulae are true in the interpretation. Let G be a set of formulae and F a formula, we say that F is a logical consequence of G , if F is true in all models of G . This is denoted by $G \models F$.

Let $(\{R_1, R_2, \dots, R_n\}, IC)$ be a relational scheme and a database state $r = \{r_1, r_2, \dots, r_n\}$, it is possible to establish an easy connection between the data model approach and the concept of interpretation. First of all, we have to define the first order language : the set of constants (is the set obtained from the union of the underlying domains of all attributes that occur in the relational schema, for each relation name R_i we assign a predicate R_i (same name for simplicity), for each independant attribute domain we relate an unary predicate β_i . Then, if we assign for every predicate R_i an instance r_i , for every unary predicate β_i its entension $E(\beta_i)$, the pair $(r, E(\beta))$ is an obvious interpretation of the language. Furthermore $(r, E(\beta))$ is a model for the integrity constraints IC expressed as formulae if IC contains formulae such that :

$$(x_1, x_2, \dots, x_n)(R_i(x_1, x_2, \dots, x_n) \Rightarrow \beta_1(x_1) \& \beta_2(x_2) \& \dots \& \beta_n(x_n))$$

for each predicate R_i .

A query can be expressed as a formula with free variable : $\langle x.\beta : F(x) \rangle$, where $x.\beta$ denotes a sequence of typed variables $x_1.\beta_1, \dots, x_k.\beta_k$ and F a formula. The query answer is a set of k -tuples (c_1, c_2, \dots, c_k) such that :

$\beta_1(c_1), \beta_2(c_2), \dots, \beta_k(c_k)$ and $F(c_1, c_2, \dots, c_k)$ are satisfied.

The notion of query defined here above does not mention the concepts of safe formula (Ul 80), and evaluable formula (Dem82). These concepts have been introduced for considering "reasonable" query. Roughly speaking a reasonable query is a query where the range of a variable is perfectly defined.

3. A logic theory for database

The first order predicate calculus is a formal system whose objects are sentences and which has a set of axioms : the logical axioms and the inference rules. It is possible to add new axioms to built a theory T . A formula F is derivable from a set of formulae G , iff F is deducible from G and from the axioms in a finite number of steps by using the inference rules. This will be denoted $G \vdash F$.

In section 2 we have shown that a database can be seen as an interpretation of a first order language. We can consider also a database from a different viewpoint (Re 81, Ko 80), as a theory where the proper axioms of the theory are representations of all the elementary facts about the database. In this approach queries and integrity constraints are formulae to be proved.

One important result is the following : if F is a formula from a relational language which is satisfied in some interpretation, then there exists a theory T where F is deducible from T and reversely.

The interest of this result is to state precisely the proper axioms of the theory :

1. Each elementary fact in the database is an axiom, that is, if $t = (a_1, a_2, \dots, a_n)$ is a tuple related to the relation scheme R , then $R(a_1, a_2, \dots, a_n)$ is in T .
2. The domain closure axiom which says that a variable can take its values only inside the domain D .
3. The completion axioms which states that for each relation predicate the only facts known to be true are those defined in the extension of the predicate, the others are false.
4. The equality axioms are related to the equality predicate. These axioms specify the usual properties of equality : reflexivity, symmetry, transitivity, principal of substitution of equal terms.
5. The unique name axioms which state that all the individual objects can be distinguished.

Let (L, I, IC) be a relational database, where I is an interpretation and IC a set of integrity constraints expressed as formulae in the language L , then it is equivalent to consider a triple (L, T, IC) where T is a theory, which admits I as a unique model. If F is a truth formula in I , F is deducible from T , $T \vdash F$. Furthermore a query $Q = \langle x.\beta : F(x) \rangle$, where x is a free-type variable, consists of those constants c such that $T \vdash \beta(c)$ and $T \vdash F(c)$.

4. Why a formal approach ?

Logic provides a convenient formalism in which the basic concepts of databases can be studied in a rigorous and unified way (GMN 81 84)(GMN 84). The two approaches : the relational model view and the theory view give complementary views of the same problem. In the relational model view the relational scheme is an incomplete specification of an application. We need application program to define precisely the database states. At the implementation level the two approaches are very different and opposite. In the theory view each formula is coded as its own, and all the retrieve operations are defined in term of theorem to be proved. This situation is favored by the community of PROLOG systems where all the elementary facts about a database are stored as Horn formulae. On the other hand, in the model view, elementary facts are stored in logical records and retrieve operations are performed by relational operations such : join, selection,... Today there is a lot of work where researchers are looking at the integration of new mechanisms for retrieval and deduction in the same system.

Logic is also useful for other aspects of databases : specification of integrity constraints, null values and incomplete information.

CHAPTER 2

DEPENDENCIES

1. Introduction

In the design process of databases more attention has been paid to special types of constraints, called the family of data dependencies. The data dependencies play a central role in the relational database theory. The introduction of functional dependencies, multivalued dependencies and subsequent formalizations capture a significant amount of semantic information which is useful to normalize and decompose relation schemes. The decomposition of a relation is a topic that has been studied extensively because progress in logic database design has been strongly related to it. Some design methodologies for professional analyst are derived from these studies. Today, we know what its limits and usability are.

When defining a relational scheme, it is not clear what is the real meaning of the relations and it is often explained implicitly. It is the objective of data dependencies to explain how data are related to each other and what kinds of connection exists between these data ?

We shall consider a relational scheme R (Agent, Product, Price, Company). It is possible to define different meanings ; it is the purpose of the following sections.

2. Functional dependencies and multivalued dependencies

Example 2.1 : the relational scheme has the following meaning: a company has agents who sell product at a given price. If we assume that all the relation instances of R have the same property: a product has a unique price, we introduce a functional dependency denoted $P \rightarrow I$. Formally a functional dependency may be defined by a formula :

$$(2.1) \quad (a,a',p,i,i',c,c')((R(a,p,i,c) \wedge R(a',p,i',c')) \Rightarrow i = i')$$

where R is the predicate name for the relation R.

Functional dependencies play a central role in the normalization process introduced by Codd. Instead of normalization we prefer the terminology of decomposition.

According to the decomposition theory if a database designer decides to store information about the relational schema $R(A,P,I,C)$ in a unique relation, this schema suffers from the following update anomalies :

1. two different tuples of a relation may contain the same product and consequently the same price, in this case there is some redundancy in the information content.

2. if a given product is only delivered by a company, and if this product is no longer provided by the company, then the price of the product is lost.

In the decomposition theory the basic idea is to replace the initial scheme $R(A,P,I,C)$ by two schemes $R1(A,P,C)$ and $R2(P,I)$, where the information about the product prices is given by $R2$. This conceptual design approach can be formalized by the decomposition process of a relation.

Let r be a relation of the relation scheme $R(X,Y,Z)$, we say that r is decomposable if we can find two relations $r1$ and $r2$, of type XY and XZ , such that :

- $r1 = r[X,Y]$, $r2 = r[X,Z]$, this means that $r1$ and $r2$ are projections of r .
- $r = r1 * r2$, the join of $r1$ and $r2$ is r .

There exists an easy generalization of decomposition in more than two parts. This concept of decomposition is very important, because instead of storing the relation r in the database, we can store only the projections of r . The goals which are achieved by decomposition are :

- reduction of the degree of redundancy contained in a relation
- breaking down the semantic facts represented in a relation into more elementary facts.

These two goals are not really independent. All the historic developments of the relational database theory illustrates the fact that researchers are trying to determine what are the different types of constraints which may influence the decomposition. Most of the family dependencies rely upon the two equivalent propositions which give the characterization for a relation to be decomposable.

A relation r of type XYZ is decomposable iff for all $x \in r[X]$ $r[x,Y,Z] = r[x,Y] \times r[x,Z]$, where \times denote the cartesian product, $r[x,Y,Z]$ the set $\{(y,z) : (x,y,z) \in R\}$. An equivalent condition is for all $(x,y) \in r[X,Y]$, $r[x,y,Z] = r[x,Z]$.

A relation is decomposable if it satisfies a property defined on the values of the relation. This fact is not very useful since it needs a complete checking of the values contained in the relation. The importance of the family dependencies is to characterize the property expressed in terms of semantic properties. For example, if r is an instance of the relational scheme $\{X,Y,Z\}$, $\{X \rightarrow Y\}$ then r is decomposable into two relations of type (X,Y) and (X,Z) . A functional dependency implies a decomposition but the reverse is not true.

If we consider our example $R(A,P,I,C)$ with $P \rightarrow I$ then we can decompose the schema into $R1(A,P,C)$ and $R2(P,I)$. Furthermore,

we can explore what is the meaning of the association between the others attributes. A company has several agents, then $C \not\rightarrow A$, but the knowledge about a company determines the set of agents independently of the product and the price ; this is the case of a multivalued dependency denoted by $C \rightarrow\!\!\rightarrow A$.

Formally we say that a relation r of type U satisfies the multivalued dependency $X \rightarrow\!\!\rightarrow Y$, where X and Y are subparts of U , and $Z = U - XY$, if for all pair of tuples s and t in r satisfying $s[X] = t[X]$ then there exists a tuple w such that: $s[X] = t[X] = w[X]$, $w[Y] = t[Y]$ and $w[Z] = s[Z]$.

It is easy to show that the definition of a multivalued dependency is equivalent to the condition for a relation to be decomposable.

We have seen that a functional dependency can be expressed as a special formula, this is the same for a multivalued dependency; the multivalued dependency $C \rightarrow\!\!\rightarrow A$ is equivalent to the formula :

$R(a_1, p_1, i_1, c) \wedge R(a_2, p_2, i_2, c) \Rightarrow R(a_1, p_2, i_2, c)$ where all the variables are universally quantified.

Thus, our knowledge about relations on the scheme $R(A, P, I, C)$ can be more precisely described by

$(\{A, P, I, C\}, \{C \rightarrow\!\!\rightarrow A, P \rightarrow I\})$.

From the multivalued and the functional dependency every relation r of the scheme can be decomposed into three parts : $r_1 = r[CA]$, $r_2 = r[PI]$, $r_3 = r[PA]$, this means that r is the join of three relations, each relation belonging respectively to scheme $R_1(C, A)$, $R_2(P, I)$, $R_3(P, A)$. These schemes are irreducible, then they can be interpreted as the atomic meaning of the initial scheme $R(A, P, I, C)$.

One can think about the relational scheme $R(APIC)$ in a different way. If we assume that for every company : "all the products delivered by a company are sold by all the company's agents", then this property can be stated in formal terms : if r is a relation for all c in $r[c]$

$$r[c, P, A] = r[c, P] \times r[c, A].$$

This expression is the decomposition condition, not for the relation r , but for a projection of r over the attributes set A, P, C . To capture this new type of dependencies we talk about embedded multivalued dependencies or hierarchical dependencies (De 73) (De 78) (TKY 7a), the notation used is $c \rightarrow\!\!\rightarrow P|A$. If the relational scheme is defined by $R = (\{A, P, I, C\}, \{P \rightarrow I, C \rightarrow\!\!\rightarrow P|A\})$, we obtain by decomposition the same result.

3. Join dependency

Example 2.2 : we have seen that multivalued dependencies characterize exactly the decomposition process of a relation into two parts. However, multivalued dependencies are inadequate for expressing the conditions under which a decomposition into more than two parts can occur. The following interpretation of the relational scheme $R(A,P,I,C)$ is used to illustrate the concept of join dependency. As in the first case, we assume that a product has a unique price, but furthermore, we have three associations of interest : an agent sells products, a company distributes product, and an agent works for a company.

We can see each association : agent-product, product-company, company-agent as a relation which is the projection of a relation r over a set of attributes. In such a case, the relational scheme $R(A,P,I,C)$ has a special property (NI 78) expressed by the join operation :

$$r[A,P,C] = r[A,P] * r[P,C] * r[C,A].$$

For example the relation r of type A,P,C satisfies this property:

A	P	C
a2	nut	c2
al	nut	c2
al	screw	c1
al	screw	c2
a2	bolt	c2

A join dependency will be denoted by $*[AP,AC,PC]$, in the present case we say that it is a partial join dependency since it is not defined over the whole set of attributes. We shall see later that it is also a cyclic dependency because the association graph has a cycle.

More formally, we can define a join dependency as a sentence, denoted $*[x_1, x_2, \dots, x_k]$, where x_i are parts of an attribute set U , and such that $\cup x_i = U$, if for every relation r of type U , r is the join of the projections $r[x_1], r[x_2], \dots, r[x_k]$. A relation r of type U satisfies this property if and only if for each tuple t of r there exists tuples w_1, w_2, \dots, w_k of r (not necessarily distinguished) such that :

$$w_i[x_i] = t[x_i] \text{ for } i = 1, 2, \dots, k.$$

A partial join dependency is a join dependency over a set U' of attributes contained in U .

If the relational scheme R is defined by

$R = (\{A,P,I,C\}, \{P \rightarrow I, *[AP,AC,PC]\})$ it is easy to deduce that every relation r is decomposable into four parts. $R1(PI)$,

$R2(A,P)$, $R3(A,C)$, $R4(P,C)$. In fact, the interpretation of a relation r is given by :

$\{(a,p,i,c) : i \text{ "is the price of" } p \text{ and } a \text{ "sells" } p \text{ and } a \text{ "works for" } c \text{ and } c \text{ "distributes" } p\}$.

If we introduce the predicate $P1$, $P2$, $P3$ and $P4$ respectively for "is the price", "sells", "works for", "distributes", the relation is of the form

$$(2.4) \quad \{(a,p,i,c) : P1(p,i) \& P2(a,p) \& P3(a,c) \& P4(c,p)\}.$$

The fact that a relation r of type A,P,I,C is of the form (2.4) is a severe constraint for some predicate $P1$, $P2$, $P3$ and $P4$, but we have the real meaning of a relation in terms of basic elementary fact. Before we leave the join dependencies, let us note, that a join dependency can be written as a logical formula. The join dependency $*[AP, AC, PC]$ can be expressed as

$$(2.5) \quad (a,al,p,pl,c,cl, il,i2,i3)((R(a,p,il,cl) \& R(a,pl,i2,c) \& R(al,p,i3,c)) \Rightarrow \exists i R(a,p,i,c)).$$

In the formula we use existential quantifier since it is a partial join dependency.

4. Inclusion dependencies

Example 2.3 : so far, all the dependencies we have studied have the property of being defined over a unique relation. The class of inclusion dependencies is a multi relation property (Da 81, Li 81, CFP 84).

For example for a relation r of type A,P,I,C , a tuple (a,p,i,c) cannot be stored in the database until we can guarantee that a is an agent, p is a product and c is a company. This kind of properties is frequently used in database system.

Thus, we need three unary relation $R1(A)$, $R2(P)$, $R3(C)$, where each relation contains respectively the list of all agents, products and companies. The constraints can be written as :

$$(2.6) \quad \begin{aligned} R[A] &\subseteq R1(A) \\ R[P] &\subseteq R2(P) \\ R[C] &\subseteq R3(C) \end{aligned}$$

In general an inclusion dependency is of the form

$R[X'] \subseteq S[Y']$ where R and S are relation scheme over sets X and Y of attributes, and $X' \subseteq X$ and $Y' \subseteq Y$. Let r and s be relations of type X and Y . An inclusion dependency holds if for each tuple t of r , there exists a tuple w of s such that $t[X'] = w[Y']$.

In the network or hierarchical model, they are implemented most of the time in an easy way inside the logical structure of the

database. In relational system we need to specify this kind of constraint between relations.

Let $R(ABC)$ and $S(CDE)$ be two relation schemes : the inclusion dependency $R[AB] \subseteq S[CE]$ can be written as a sentence in a first order language :

$$(2.7) \quad (abc)(R(abc) \Rightarrow (Ed) S(adb)).$$

The examples that we have presented illustrate the fact that dependencies capture the basic properties of association between data. Moreover the decomposition process modelizes database design in such a way that the output is a consequence of the input data dependencies. Figure 2.1 is a graphical representation of our knowledge about the real logical structure presented in the examples 2.1, 2.2 and 2.3. A relational scheme can be represented by an hypergraph where nodes are attributes and edges are sets of attributes.

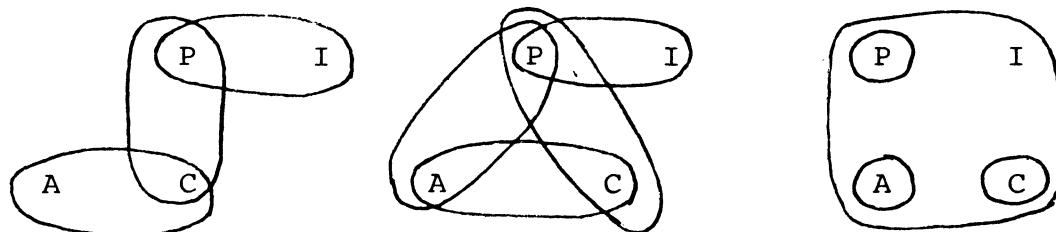


Figure 2.1 : graphical representation of a relational scheme.

5. General expression for dependencies

Dependencies can be written as first order sentences where we consider :

- predicate relations of the form $R(\dots)$,
- the equality predicate, $x=y$ where x and y are variables,
- an atomic formula is a predicate relation or an equality predicate

A dependency formula is of the form (Fa 84) :

$$(2.7) \quad (x_1 \dots x_m)(A_1 \wedge \dots \wedge A_n) \Rightarrow (E y_1 \dots y_k) \\ (B_1 \wedge \dots \wedge B_r)$$

where the A_i 's are predicate relation and the B_i 's atomic formulae, the x_i 's and y_i 's are variables. In the case $k=0$ there are none existential quantifiers.

One class of dependency formulas has been studied extensively because they are related to join dependencies, it is the class of implicative dependencies (BV 81)(SU 81) which are special Horn formulae with a unique predicate P . An Horn formula :

$$(2.8) \quad (P(x_{11}, \dots, x_{1n}) \wedge \dots \wedge P(x_{m1}, \dots, x_{mn})) \Rightarrow \\ P(x_{01}, \dots, x_{0n})$$

is an implicative dependency, if $x_{ij} = x_{kl}$ implies $j=1$, ($0 \leq i, k \leq m, l \leq j, l \leq n$). An implicative dependency can be represented by a tableau where the entries are formed from the symbols taken in the formula.

Entries from 1 to m are the conditional entries, and the last entry is the conclusion. The tableau associated with the formula 2.8 is :

$$\begin{array}{c} x_{11} \dots x_{1n} \\ x_{21} \dots x_{2n} \\ \vdots \\ x_{m1} \dots x_{mn} \\ \hline x_{01} & x_{0n} \end{array}$$

According to the definition if a variable appears in different entries of the tableau then this variable is always in the same column.

A full implicative dependency is an implicative dependency if the conclusion variables are contained in the condition variables. An implicative dependency is called reduced if for all $i, j (i \neq j) : x_j \cap x_0 \subseteq x_i \cap x_0$ (x_i denotes the entry numbered i), and is called join implicative dependency, if for all $i, j (0 < i < j \leq m) (x_j - x_0) \cap (x_i - x_0) = \emptyset$. A join implicative dependency is equivalent to a join dependency. For example the join dependency

* [AB,BCD,CDEF,BCE] is represented by the tableau.

A	B	C	D	E	F
a	b	x_1	x_2	x_3	x_4
x_5	b	c	d	x_6	x_7
x_8	x_9	c	d	e	f
x_{10}	b	c	x_{11}	e	x_{12}
<hr/>					
a	b	c	d	e	f

These kind of dependencies were introduced to deal with the issue of a complete axiomatization, it is the purpose of chapter 3.

CHAPTER 3

THE IMPLICATION PROBLEM

1. General situation

Given a relational scheme $R = (\{R_1, R_2, \dots, R_n\}, F)$ where the R_i 's are relation schemes and F a set of constraints, we say that a constraint f is a logical consequence of F , $F \models f$, if all interpretations that are models for F are the same as for f , or in other terms $SAT(F) \subseteq SAT(f)$.

In examples given in chapter 2 we have illustrated different situations where data dependencies are very useful to capture semantic properties about data, and we have demonstrated how the decomposition process breaks down a relation into elementary facts. In the decomposition process the projection operation of the relational algebra is the basic one, but a generalization can be done in the following way.

To a relational scheme $R = (\{R_1, R_2, \dots, R_n\}, F)$ we shall associate through a mapping μ a relational schema S , such that S is the image of R by μ . The mapping can be defined by considering a sequence of elementary mapping μ_i , $\mu = (\mu_1, \mu_2, \dots, \mu_k)$. Each μ_i can be expressed by an algebraic relational sentence built up from relation names R_1, R_2, \dots, R_n . Formally we denote by $S_i = \mu_i(R_1, R_2, \dots, R_n)$ the relation scheme obtained, and an instance of S_i is $s_i = \mu_i(r_1, r_2, \dots, r_n)$ where the r_i 's are instances of the R_i 's. The scheme S is composed of $(\{S_1, S_2, \dots, S_k\}, \mu(F))$ and it can be thought as the image of R , (figure 3.1).

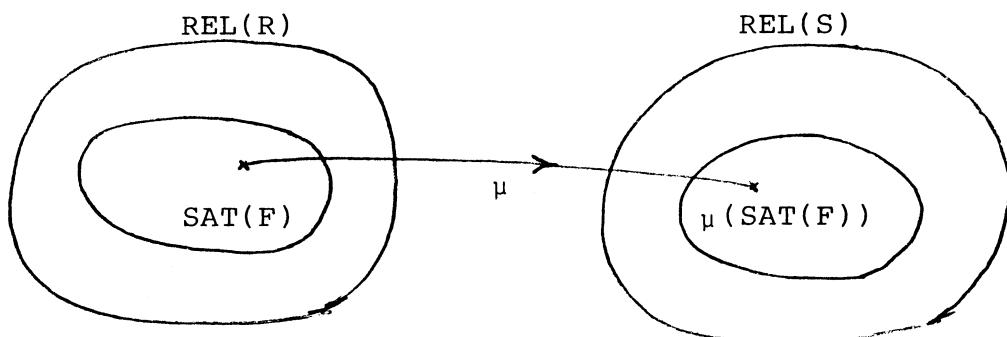


Figure 3.1 : mapping scheme.

The mapping μ transforms the set of constraints F into a set of constraints $\mu(F)$ such that :

$$SAT(\mu(F)) = \{\mu(r) : r \in SAT(F)\} = \mu(SAT(F)).$$

In the definition of R , the weakest constraint, denoted \emptyset satisfies $SAT(\emptyset) = REL(R)$. A constraint over S is the consequence of the mapping μ . We shall say that g is a constraint over S

if g is a logical consequence of $\mu(\emptyset)$, that is $\mu(\emptyset) \vdash g$, or equivalently $SAT(\mu(\emptyset)) \subseteq SAT(g)$, or $(REL(S)) \subseteq SAT(g)$. It is clear that $\mu(\emptyset)$ is the strongest constraint over S . The studies about dependencies have two purposes :

- (i) how is it possible, inside a scheme, to know that a constraint is a logical consequence of a set of constraints?
- (ii) how is a constraint modified or preserved by a mapping?

Theoretical results have been obtained in special cases, where the mapping is easily expressed by projections or joins.

1st case : the relational scheme R is reduced to a unique relation name, R itself, and the mapping is composed of k projections, $\mu = (\mu_1, \mu_2, \dots, \mu_k)$, such that $\mu_i(r) = r[X_i]$, where r is an instance of R and X_i is a part of $\delta(R)$. The image of R is $S = (S_1, S_2, \dots, S_k, \mu(F))$, where $\delta(S_i) = X_i$, for $1 \leq i \leq k$.

A database state (s_1, s_2, \dots, s_k) satisfies the universal relation assumption if there exists a relation r such that the type of r is the union of $\delta(S_i)$, and the s_i 's are the projections of r , $s_i = r[X_i]$.

It is clear that in the first case the scheme S obeys the universal relation assumption and it is the strongest constraint.

A mapping is only useful when no loss of information occurs. More precisely, we must be able to reconstruct r from (s_1, s_2, \dots, s_k) , (figure 3.1). For that, it is sufficient that μ is injective for the relations r that obey F . If the mapping μ is bijective it is called a faithful representation.

Under the assumptions that μ is faithful and F is composed of functional dependencies, the inverse of μ , which is the reconstruction map, is the join and S is a decomposition of R . Generalization of this property can be obtained for multivalued dependencies and total join dependencies (BR 79) (MMSU 80). The notion of faithful representation is too strong a condition since in many practical situations the condition is not satisfied when the user makes updates.

2d case : let R and S be relational scheme defined by $(\{R_1, R_2, \dots, R_n\}, F)$ and $(S, \mu(F))$, where $\delta(S) = \bigcup \delta(R_i) = \bigcup X_i$, the mapping μ is the join, that is $\mu(r_1, r_2, \dots, r_n) = r_1 * r_2 * \dots * r_n = s$. It is clear that S obeys the join dependency :

$$* [X_1, X_2, \dots, X_n]$$

and it is the strongest constraint.

2. Axiomatization for dependencies

A lot of effort has been devoted to the study of axiomatization for dependencies. Axiomatization is important because it give us means for determining the whole set of dependencies contained

in a relational scheme. Properties of dependencies can be seen as inference rules, that is rules that deal with the implication of new dependencies from a given set F of initial dependencies.

A formal system for a family of dependencies is composed of axioms and inference rules. A derivation of a dependency f from F , $F \models f$, is a sequence of dependencies $f_1, f_2, \dots, f_n = f$, where each f_i is either an axiom, or a member of F , or is derived from the preceding dependencies in the sequence. We shall say that an inference system is sound if $F \vdash f$ implies $F \models f$. It is complete if $F \models f$ implies $F \vdash f$.

Many sound and complete systems for dependencies have been studied. The first one has been obtained by Armstrong for the functional dependencies. Where he proved that the properties of reflexivity, augmentation and transitivity constitute a complete set of inference rules. Similar results have been obtained for multivalued dependencies, full join dependencies, and inclusion dependencies. Below, one can find inference systems for homogeneous family of dependencies.

The problem of determining a complete set of inference rule for embedded dependencies has been considered as one of the more difficult in the database theory. The first negative result has been obtained in 1982 where it has been proved that it is not possible to find such a system for the embedded join dependencies (PP 80). The work on inclusion dependencies revealed that the concept of logical consequence for dependencies under a finite interpretation must not be confused with an infinite interpretation.

Functional dependency (Ar 74)

FD1 (reflexivity) : if $Y \subseteq X$ then $X \rightarrow Y$

FD2 (augmentation) : if $Z \subseteq W$ and $X \rightarrow Y$ then $XW \rightarrow YZ$

FD3 (transitivity) : if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

Multivalued dependency (BFH 77)

MVD0 (complementation) : $X \rightarrow\rightarrow Y$ iff $X \rightarrow\rightarrow U-Y$

MVD1 (reflexivity) : if $Y \subseteq X$ then $X \rightarrow\rightarrow Y$

MVD2 (augmentation) : if $Z \subseteq W$ and $X \rightarrow\rightarrow Y$ then $XW \rightarrow\rightarrow YZ$

MVD3 (transitivity) : if $X \rightarrow\rightarrow Y$ and $Y \rightarrow\rightarrow Z$ then $X \rightarrow\rightarrow Z$

Multivalued dependency and functional dependency (BFH 77)

FD-MVD1 : if $X \rightarrow Y$ then $X \rightarrow\rightarrow Y$

FD-MVD2 : if $X \rightarrow\rightarrow Z$ and $Y \rightarrow\rightarrow W$ where $W \subset Z$ and $Y \cap Z = \emptyset$ then $X \rightarrow\rightarrow W$.

Full join dependency (AD 80 for binaryjoin) (Th 84)

JD1 (reflexivity) : * [U]

JD2 (augmentation) : * $[X_1, \dots, X_k]$ implies * $[Y_1, \dots, Y_k]$
if for all $1 \leq i \leq k$ there exists
 $1 \leq j \leq m$ such that $X_i \subseteq Y_j$

JD3 (decomposition) : if $*[x_1, \dots, x_k]$ and $*[y_1, \dots, y_m]$
then $*[z_1, \dots, z_k, y_2, y_3, \dots, y_m]$ where
 $z_i = (x_i \cap (x_1 \cup \dots \cup x_{i-1} \cup x_{i+1}) \dots \cup x_k) \cup (x_i \cap y_i)$
for $1 \leq i \leq k$.

These results are only valid for homogeneous family of dependencies.
In the real world this situation is not really satisfied. In such
a case a complete inference system has been only proved for
functional dependencies and multivalued dependencies.

3. Degrees of cyclicity for join dependency

A full join dependency $*[x_1, x_2, \dots, x_k]$ belongs to the class JD-k if it is composed of k parts. We shall say that a full join dependency D has a degree of cyclicity equal to n if there exists a set P of full join dependencies belonging to the class JD-n+2 such that $P \vdash D$ and $D \vdash P$ (Th 84).

A full join dependency is acyclic when the degree is zero.
From the definition an acyclic dependency is equivalent to a set of binary join dependencies, which is itself equivalent to a set of multivalued dependencies (FMU 82).

The concept of degree of cyclicity for join dependencies has been introduced in different way by Fagin (Fa 83) and different characterizations have been given in terms of properties of hypergraph. These definitions are generalization of the notion of Berge's cycle. For testing that a join dependency is acyclic there exists a simple algorithm (Gr 79).

Example 3.1

- * [AB, AC, BC] is of degree 1
- * [AB, CD, ACE, BDF, EF] is of degree 3.

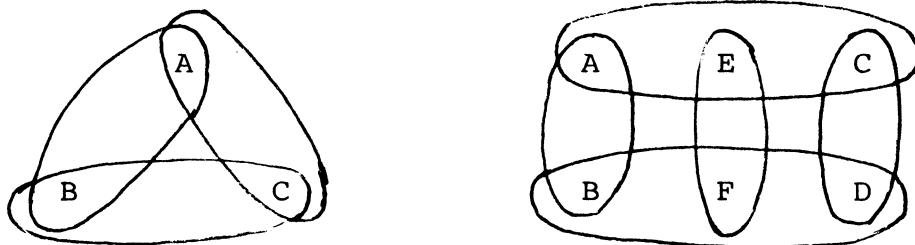


Figure 3.2 : hypergraphs related to join dependencies

4. Algorithmic manipulations for dependencies

Due to the apparent complexity of the inferential structure, good algorithms for manipulating dependencies have not accompanied all the theoretical developments.

Among algorithmic manipulations we are considering five basic problems : dependency closure and covers, dependency membership,

keys of a relation scheme, losslessness, degree of cyclicity for a join dependency.

The dependency closure is the set of all dependencies implied by a set of dependencies. A cover for a set of dependencies is a non-redundant set of dependencies which generates the closure of that set. The concept of cover is fundamental in database design using only functional dependencies, since a cover is a synthesis of all the functional dependencies. The losslessness property for a set $\{X_1, X_2, \dots, X_k\}$ is equivalent to say that $*[X_1, X_2, \dots, X_k]$ is implied by a set of dependencies. This problem is related with the decomposition process.

Most of these algorithmic manipulations are based on two techniques : the boolean techniques and the tableau techniques and some of them are applied only to dependencies for which we know a complete inference system. The advantage of the boolean techniques is to provide efficient manipulations for the inferring rules in a formal system of dependencies. It is possible to establish an isomorphism between a fragment of propositional logic and a set of functional and multivalued dependencies (SDPF 81).

The tableau techniques (ABU 77) allow us to check if a dependency is implied by a set of dependencies but it does not tell us how to generate new dependencies. The concept of implication dependency introduced in section 2.5 is strongly connected with tableaux techniques.

All the theoretical studies are useful for at least three reasons:

1. It is possible to state some database design techniques on theoretical grounds. The concept of data dependencies characterizes the relationships between the objects of the real world in an application. The consistency and redundancy of a set of data dependencies can be tested precisely before deriving by a synthesis or decomposition process a relational scheme.
2. The approach presented here has the drawback that most of the theoretical results rely upon the universal relation assumption. Nevertheless, this has been useful since many researchers have studied how to use this concept for new kinds of database systems. Chapter 4 is devoted to those problems.
3. The formal inference systems developed for dependencies are special cases of mechanisms used in expert systems. In some applications of expert systems one can find problems which can be modeled with the same structure of dependencies. For example, in the university environment the prerequisite rules for a unit of course can be expressed by a functional dependency model. There exists probably analogous situations where a data dependency model could be used in an efficient way to model expert applications.

CHAPTER 4

INTERPRETATIONS AND UTILISATIONS OF THE UNIVERSAL RELATION

1. The different interpretation

The universal relation assumption has been introduced in many works where the motivations did not appear clearly at the beginning.

One of the first motivation is related to the necessity of deriving meaningful data dependencies. For example, consider a relational scheme over the attribute : Student, Unit of Course, and Teacher, where $R = (R_1, R_2, F)$ with $\partial(R_1) = \{S, U, T\}$ and $\partial(R_2) = \{S, T\}$. The relation scheme R_1 means that a student takes a course given by a teacher, while R_2 means that a student is supervised by a teacher. If the constraints F are of the form: a student takes only one course, a course has only one teacher, we can write $S \rightarrow U \rightarrow T$ in R_1 , and $S \rightarrow T$ in R_2 .

It is clear that the dependency $S \rightarrow T$ in R_2 is not redundant with $S \rightarrow T$ in R_1 obtained by transitivity. In such a case the relationship between the attributes S and T has two different meanings and the universal relation assumption is not satisfied. The only way to cure this problem is to introduce new names for the attributes. Instead of T in R_2 , we can define a new name for the supervisor (V). We must say also that the supervisors are teachers which can be expressed by an inclusion dependency.

In database scheme design, the goal is to built a database scheme which can be explained from the inputs. The implicit framework is the following sequence :

Input Scheme \longrightarrow Universal Scheme \longrightarrow Output Scheme

where the universal scheme is a central view where all the requirement analysis and constraints have been integrated. Every relation of a universal scheme is necessary redundant. For many reasons : redundancy and update anomalies it is useful to break down a universal relation into small parts as explained earlier. The principal activity in database design is the decomposition of the universal into a database scheme that has some properties expressed in normal forms.

The third motivation is to offer to users an easy manipulation language. One of the main goal introduced by Codd with the relational model is to provide a high level language where programmers specify what they want without telling the access paths to the data. First order languages have this property, but nevertheless programmers must specify the logical connections between relations. Systems built with the universal relation assumption can remove this fact and provide an easy programming interface. Nevertheless, it introduces new kinds of problems.

For example, consider a relational scheme over two relation schemes PC (Parent, Child) and CT (Child, Toy). If we are interested in the relationship between parent and toy, we can built a relation

over PT by writing $[PT] = (PC * CT) [PT]$. If a universal scheme exists over PTC, to obtain the same result we need only a projection over the attributes PT. This is the main advantage of the universal relation where the easy requests can be expressed by projection or selection operations. We don't have any guarantee that the result delivered by the universal scheme has the correct meaning that we assign to the request. In fact, a universal relation is always a view of a conceptual database, for example the universal relation PTC is equal to PC * CT. Thus, a query like "give me information about parent and toy" is interpreted as a projection of the universal relation if we have in mind that we want a toy related to a parent such that there exists a child connected with the toy and the parent. In this case the result of the universal relation is correct. But if we assume that the childs can be considered as parents, then it is not obvious when building a relation over PT that we want a toy related to the parent through a child or a toy related to the parent of the parent, etc... . The user might have in mind something other than the simplest relationship. Another example illustrates a different kind of ambiguity with the universal relation.

Example 4.1. Let R be a relational scheme,
 $R = (R_1(\text{Customer}, \text{Account}), R_2(\text{Customer}, \text{Loan}), R_3(\text{Loan}, \text{Bank}), R_4(\text{Account}, \text{Bank}))$ where R_1, R_2, R_3 and R_4 can be interpreted as: a customer has an account, a customer has a loan, a loan is in a bank, an account is held by a bank. If we define a relation scheme $S(C, A, L, B)$ where every instance s of S is the join of r_1, r_2, r_3 and r_4 , $s = r_1 * r_2 * r_3 * r_4$.

The relation s is a universal relation if the projections of s over CA, CL, LB and AB are equal to r_1, r_2, r_3 and r_4 . If we are interested in the relationship between a bank and a customer there are two possible connections (figure 4.1).

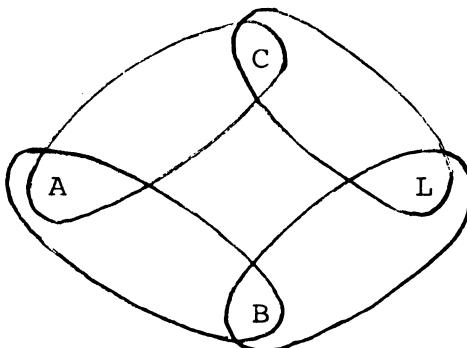


Figure 4.1 : the customer-bank-loan-account scheme.

- a bank is connected to a customer through an account,
- a bank is connected to a customer through a loan.

An user querying a universal relation about the relationship between bank and customer sends to the system a query of the form $s[BC]$. A pair (bc) is returned if the customer c holds both a loan an account in the bank b. If the user had in mind "give the set of

pairs (b,c) such that the customer c holds a loan in the bank b or c holds an account in the bank b' then the projection of the universal relation over BC is not the correct answer. To satisfy this case the appropriate relational expression for the universal relation should be $s = (r_1 * r_4) + (r_2 * r_5)$.

The graph representing the relational schema R (figure 4.1) is cyclic, and the fact mentionned with the relationship between Bank and Customer is related to the existence of a cycle.

Before discussing and presenting some universal models, let us consider the basic assumptions illustrated in the examples presented above in this section.

1. Unicity of attribute "role"

In a universal relation each attribute plays a unique role. Thus an attribute like DATE cannot stand for birthdate, hiring date,

Before building a universal relation it is necessary to rename attributes which play different roles.

2. Unicity of relationship

This assumption can be seen as a corollary of the previous assumption. If we are interested in a relationship between the attributes X , there is only one meaning related to X which is embedded in the universal schema. The meaning of the X -relationship depends on the relational expression used to built the universal relation.

3. Union compatibility for relationships

In the example 4.1 we have seen that the relationship between the X -attributes can be evaluated by different logical connections. Which one should be used, or what combination should be considered? A universal relation favors one of these solutions, and when querying the universal relation the user should have in mind this knowledge. The solution taken by universal relation models is that all the elementary facts about the X -connections represent the same flavor, this means that all the elementary facts can be merged in the same box.

The two basic problems with universal relation models are :

1. How to built from a relational scheme

$R = (R_1, R_2, \dots, R_n, F)$ and a database state $r = (r_1, r_2, \dots, r_n)$, a representation of the universal relation which preserves the information content of the database and satisfies the constraints in F ?

2. Assuming that a universal representation exists, how is it possible to built the answer for a relationship over the attributes X ?

If the scheme R satisfies the universal relation assumption, then the representation is the universal relation defined by $s = r_1 * \dots * r_n$, and every request over the association X will be evaluated to $[X] = s[X]$. The main difficulty is to know if

a database state obeys the universal relation assumption. Testing this property is NP-complete (HLY 80). If the relational schema R is acyclic the problem is simplified since we need only to take relations two by two (BFMY 83).

The universal relation assumption is a very strong assumption and it is not possible to handle the case where an update does not maintain the assumption. We outline here one approach to remove this problem, called the weak universal relation approach.

2. The weak universal relation assumption

Let $R = (R_1, R_2, \dots, R_n, F)$ be a relational scheme, where $\partial(R_i) = X_i$ for $1 \leq i \leq n$, we shall say that a database state $r = (r_1, r_2, \dots, r_n)$ obeys the weak universal relation assumption (Ho 82) if there exists a relation u of type $U = \cup X_i$, such that:

- (i) $r_i \subseteq u[X_i]$, for $1 \leq i \leq n$,
- (ii) the relation u satisfies the constraints F .

From this viewpoint, a database is seen as a partial specification of the universal relation u . The definition of u is existential in nature and does not contain any operational procedure to built such a relation. Furthermore, there might exist an infinite set of relations u which satisfy the assumption. Among this infinite set it is impossible to know which of the possible ones actually represent the truth. For this reason, the state of a relationship X between attributes, denoted $[X(F)]$, is therefore taken to be:

$$[X(F)] = \{u[X] : u \text{ is a weak universal relation}\},$$

$[X(F)]$ is defined from the information contained in tuples t over X such that there exists a tuple t' belonging to a weak universal relation and $t[X] = t'[X]$. From another point of view, we can say that the answer $[X(F)]$ is deduced from the database state $r = (r_1, r_2, \dots, r_n)$ and from the constraints F seen as inference rules.

It is for this reason that the notation used for the answer $[X(F)]$ is with respect to F . A weak universal relation can be seen as a deductive approach to databases.

The major problem is to build a single relation, which may contain incomplete information, and which embodies the information present in all weak relations. This relation is called the representative instance of the weak universal relation assumption.

How to build a representative relation (Ho 82) (Sa 81) ?

The process is divided into four steps as follows :

1. We begin to build a relation of type U by picking successively a relation r_i in the database, and for each tuple t in r_i , we insert t in the relation according to the place of each

t -attribute, and for each attribute in $U-X_i$ we insert a null-value, denoted # followed by a marked place subscript i.

2. We generate new tuples by the tableau technique according to the constraints in F. In practice, in this approach the set of constraints is a set of data dependencies. In the case of functional dependencies we equate null symbols with non null symbols, if possible ; if we equate null symbols we replace the null symbol having the highest subscript by the lowest.
3. For certain kinds of dependencies, in particular embedded dependencies, we introduce new nulls and the process may not terminate.
4. The process ends in two cases : the chase terminates correctly without infinite loop or we are forced to equate two distinct non-null symbols. In this case, there is no representative instance of all the weak universal relations.

If we succeed in the process, the fundamental property about the representative relation is that it always produces the intersection of all weak universal relations. The answer $[X(F)]$ is therefore taken to be : $\text{rep}[X]$, where rep denotes the representative relation.

Example 4.2 : let R be relational scheme defined by

$R = (\{\text{CS}, \text{CSG}, \text{CP}\}, \{*, [\text{CSP}, \text{CP}], * [\text{CS}, \text{CP}]\})$

where C,S,G,P stands for Class, Student, Grade and Professor, the meaning of the relationship is respectively : a student is attending a class, a student in a class has a grade, a class has a professor. A database state is given by the following tables of values :

C	S	C	S	G	C	P
math	john	english	john	a	math	ronald
english	bill				english	mike

the representative instance is :

C	S	G	P
math	john	# 1	# 2
english	bill	# 3	# 4
english	john	a	# 5
math	# 6	# 7	ronald
english	# 8	# 9	mike

math	john	# 10	ronald
english	bill	# 11	mike
english	john	a	mike

If we introduce between tuples a partial order relation defined as follows : $t \leq t'$ if for all attribute A in U we have $t[A] = t'[A]$ or $t[A] = \text{null}$. Consequently for relations $r \leq r'$, if $(t) \in r$, $(Et') \in r'$ such that $t \leq t'$. With this partial order relation we can limit the representative instance to only maximal tuples.

math	john	# 10	ronald
english	bill	# 11	mike
english	john	a	mike

How to calculate $\text{rep}[X]$?

The relation rep may be obtained in exponential time since the chase is of the same nature. Then the evaluation of $[X(F)]$ from $\text{rep}[X]$ is not necessarily an operational solution.

The other approach taken is to look for a relational expression E whose operands are relations schemes R_1, R_2, \dots, R_n and whose operations are : join, projection, union, difference, selection, such that :

$$\begin{aligned} & E(R_1, R_2, \dots, R_n) = \text{rep}[X] \\ \text{or} \quad & E(R_1, R_2, \dots, R_n) \subseteq \text{rep}[X] \end{aligned}$$

If this happen we can say that the expression E simulates the evaluation of $\text{rep}[X]$. To obtain such a result the research works have explored different properties of the expression E :

- monotonicity
- losslessness.

Example 4.3. We illustrate the losslessness property on the example 2.2 presented at the beginning. Let $(\{\text{APIC}\}, \{\star[\text{PI}, \text{AP}, \text{AC}, \text{PC}] \rightarrow \text{I}\})$ be a relation scheme and $X = [\text{IA}]$ the relationship between Agent and Price. The algebra expression

$$(\text{PIC} \star \text{PA} \star \text{AC}) \quad [\text{IA}]$$

is lossless.

To test the property we proceed as follows : we built a tableau expression where the first t_i 's are the hypothesis rows and the last is the conclusion row. The components of these rows are abstract symbols representing attribute variable like in an implicative dependency (see section 2.5). The expression will be lossless if by some derivation rules the conclusion is obtained

from the hypothesis. The following tableau is related to the expression:

A	P	I	C	
	p	i	c	(PIC)
a	p			(PA)
	a		c	(AC)
		a	i	

Informally, these tableau can be read as : the pair (a,i) is in the answer if there exists three tuples of the form (p,i,c), (a,p) and (a,c) belonging respectively to PIC, PA, and AC. If we apply the join dependency *[PI, AP, AC, PC] we derive a tuple (a,p,i,c) and by projection (a,i). Then (a,i) is derived from the hypothesis, thus the expression is lossless.

The expression (PIC*PA) [IA] leads to the same result as shown by the tableau :

A	P	I	C	
	p	i	c	
a	p			
	a		i	

in these case the sequence of derivation is not the same. The functional dependency $P \rightarrow I$ implies that the entry (ap) generates (a p i), and by projection (a,i).

The example demonstrates that different expressions lead to the same evaluations but the computational complexity is not the same since the number of joins to compute is not the same.

An expression E is monotone, if $r_i \subseteq r_i'$ ($i=1,n$), then $E(r_1, r_2, \dots, r_n) \subseteq E(r'_1, r'_2, \dots, r'_n)$, where the r_i 's and the r'_i 's are instances of the relation schemes R_1, R_2, \dots, R_n . The basic algebra operations : join, selection, projection and union are monotone.

Under the properties of monotonicity and losslessness we can obtain the following fundamental result (MUV 84) : let E be an expression that is monotone and lossless with respect to some set of implicational dependencies F and produces a relation over X, then $E \subseteq \text{rep}[X]$. The equality can be obtained by some other characterizations.

3. Maximal objects

Example 4.1 has shown intuitively that when the scheme is not acyclic, the interpretation of the relationship between attributes can be confused ; in general several access paths are suitable for computing this connection. To avoid this difficulty one can introduce the notion of maximal objects (MU 80). A maximal object is a subset of relation schemes which inside the maximal object we are able to navigate without ambiguity, or one can say that the scheme of a maximal object is acyclic. The set of all the maximal objects covers the database scheme. In example 4.1 we can consider two maximal objects :

$$M_1 = \{AC, AB\}$$

$$M_2 = \{CL, LB\}$$

For each maximal object we take the join of all the relations belonging to it. And after we extend the result by introducing null values to all attributes which are not in the maximal object. The result obtained is a relation of type U, denoted $u(M)$. The representative instance in the maximal objects model will be the union of the relations of the form $u(M)$ for all M . In our example $u(M_1) = AC * AB$, $u(M_2) = CL * LB$, and consequently $u = u(M_1) + u(M_2)$.

A query over the relationship between the X-attributes will be evaluate to :

$$[X] = \bigcup_{M \supseteq X} u(M) [X]$$

If the set X of attributes is not contained in a maximal object then there is no answer. In this case one can consider that some attributes are so semantically distant that no connection among them can reflect a normal situation.

Several systems support a universal view of data based on this model : "SYSTEM U" (KU 80) and "PITS" (MRSSW 82). In practice, the query manipulation languages related to the universal view are very simplified. They permit queries with several tuple variables, each of which may range over a separate copy of the universal relation, where the two basic operations are selections and projections.

4. A virtual universal relation

This approach has been introduced by (BB 83) and extended in (Ve 83). In some ways, it is similar to the preceeding ones presented here. The goal is to built a representative instance of the universal relation which is called the virtual universal relation. The process for defining the virtual relation and the basic parameters for the data model are different. The virtual universal data model is based on the following basic concepts:

- objects are sets of attributes and represent elementary facts or units of update (Sc 80),

- associations are connections between objects,
- irredundant scheme.

We shall illustrate this model with an example which is the program of movies in a city for a given week. One can consider the set of attributes :

- A : actor,
- M : movie,
- T : movie theater,
- P : price,

- S : schedule.

An object is a subset of attributes related to an elementary fact. The object (MTP) characterizes facts like : "the price to pay to see the movie "the specialists" in the movie theater "the gaumont" is 30 francs". Also a fact like "the movie the specialists is presented in the gaumont theater" is associated with an object. Every application is modelled by a set of objects, intuitively for our example we have :

OBJECT = { AM, MT, MTP, MTS, MS, TP, MTPS, AMT, AMTP, AMTPS }

AT is not an object since there is no elementary sentence where A and T are related. The relationship between these two attributes implies the existence of a movie : there exists a movie m such that the actor a plays in the movie m and the movie m is played in the movie theater t. If X is an object, and Y is included in X, then Y is not necessarily an object, for example, MTS is an object and TS is not an object.

The set of objects defines the set of elementary facts which are true in the database. A fact x over an object X is a tuple, that is an application between x and the values taken in the domain of X. Certain facts may be considered as elementary and others are derived by inference rules from the elementary facts. If the actor "giraudeau" plays in the movie "the specialists" and "the specialists" is played in the movie theater "the gaumont" we are able to deduce that "the gaumont" plays the specialists movie where giraudeau is an actor of the movie".

A relation, called JOIN, defines the connections that the user considers as acceptable in its application (Ve 83). An element of JOIN is a pair of objects (X,Y) which can join to derive a new object. The relation JOIN has the following properties :

- symmetry
- if $X \subseteq Y$ then $(X, Y) \in \text{JOIN}$
- if $(X, Y) \in \text{JOIN}$ there XY is an object,
- if $(X, Y) \in \text{JOIN}$ and $Y \subseteq Z$ and Z is an object then $(X, Z) \in \text{JOIN}$.

For example, the relation JOIN contains at least two elements {(AM,MT), (MTP,MTS)}. We shall note that MS and AM are not connected

since the connection needs to know the attribute T. The relation JOIN is tailored by the database designer and establishes all the connections which are semantically useful for the user. In (BB 85) it is assumed that two objects are connected until their intersection is not empty. The relation JOIN permits these case. This can be the case in our example if we add the attribute D (day) by saying that the movie schedule for one day is the same for all days of the week, thus the pair (D,AMTPS) is an element of the JOIN relation. There is some situations where the model needs to be improved. Assume that the elementary facts over MTP can be obtained either from MTP directly, or from the objects MT and TP. This case is related to the fact that some movies theater have a unique price for all the movies and some others have a different price for each movie. Note that we use the union compatibility assumption for the relationship over MTP.

One can conceive a model where an association between the attributes X will be defined from objects Y_1, Y_2, \dots, Y_k and by a relational expression. A virtual view of the user will be built (Za 77).

In the various approaches presented, the basic idea is to built a representation instance of the universal relation as a table of values where the information contained in the table is composed of elementary facts or derived facts using the join operation or join dependencies.

The fact that we use only the join operations is a limitation for the system. In practice some other kinds of deduction can be made. Assume that we have the following facts :

- the boat b is tagged to the peer p in the city c,
- the water depth at the peer p is t

then we are able to conclude : the draught of water of the boat b is less than t.

Most of the models in the universal relation do not modelize this situation.

Let x and y be two elementary facts, respectively over the objects X and Y. We shall say that x and y are joinable and produces a new fact $x*y$ defined by :

$(x*y) [X] = x$ and $(x*y) [Y] = y$ if the pair of objects (X,Y) belongs to the relation JOIN.

A virtual relation r will be constructed by the following process: all the elementary facts will be inserted in r as tuples, eventually using a null values denoted by a blank. The closure of r, r^* , contains all the elementary facts that we can deduce from r according to the JOIN relation.

A	M	T	P	S
delon	"notre histoire"			
baye	"notre histoire"			
	"notre histoire"	gaumont		
	"amadeus"	club		14.00
	"amadeus"	club		25
	"notre histoire"	gaumont		15.00
	"notre histoire"	gaumont		30
delon	"notre histoire"	gaumont	30	15.00
baye	"notre histoire"	gaumont	30	15.00

The two bottom lines in the table are derived. If we use the partial order relation between tuples defined in example 4.2 we can restrict ourself to the maximal objects, denoted $\max(r^*)$.

Another important feature of the virtual universal relation model is the concept of generator scheme. A generator scheme is a set of objects which satisfies the following properties :

(i)- S is a subset of OBJECT,

(ii)- for each object X, every facts over X can be inferred from facts over objects included into S,

An irredundant generator scheme is a generator scheme such no proper subset is a generator. The set $S = \{ AM, MT, MTP, MTS, TP, MS \}$ is an irredundant generator for the movie application. The notion of generator scheme is also related to the notion of atomic facts.

The set of atomic facts can be obtained by projecting the virtual relation r over the generator scheme, the projection will be denoted $r[S]$ and defined by :

$$r[S] = \{ t : \partial(t) \in S \text{ and } (\exists s \in r)(s \geq t) \}.$$

A relation r containing eventually null values, is said to be lossless with respect to S iff $r = (r[S])^*$. This definition is a generalization of the decomposition of a relation over a set of attributes, and we have the following proposition (Ve 83) : if r is a relation without null values, then r obeys the join dependency $[S]$, where S is a generator. The converse is false, this means that if a relation r obeys a join dependency the facts that we can deduce by the closure of r contain the facts derived by the join dependency. It is possible to find some characterizations where the lossless property is equivalent to the join dependency.

CONCLUSION

Theoretical works in database systems has been concentrated around the relational data model. The studies form a fundamental basis for the development of homogeneous capabilities. Thus, the first order languages provide an unified view for data manipulation languages and integrity constraints. On another hand, data dependencies are formal efforts to capture semantic properties of data used in database design methodologies (KYT 79, KTT 83). The theory of normal forms, which has not been presented in the paper, deals also with database design techniques and is a complement to the notions of : relation decomposition, universal relation assumption, join dependency, and acyclic scheme. Today, all these elements are major contributions of theoretical computer science to study the fundamental problem of scheme equivalence. The history of database systems has shown that data independance is one of the major goal to achieve. Relational systems are part of the evolution with non procedural data manipulation languages. Systems supporting the universal relation assumption go still further in this direction.

BIBLIOGRAPHY

- (ABU 79) Aho A. Beeri C. Ullman J., The theory of joins in relational databases, Proc. 18th Symposium on Foundations of Computer Science, 1977, Providence, ACM TODS, 43, 1979.
- (AD 80) Armstrong W. Delobel C., Decompositions and functional dependencies in relations, ACM TODS, 5.4 1980.
- (AR 74) Armstrong W., Dependency structures of data base relationship, Proc. IFIP, North Holland, 1974.
- (BFH 77) Beeri C. Fagin R. Howard H., A complete axiomatization for functional and multivalued dependencies in database relations, Proc. ACM SIGMOD, Toronto, 1977.
- (BFMY 83) Beeri C. Fagin R. Maier D. Yannakakis M., On the desirability of acyclic database schemes, JACM, 30.3, 1980.
- (BR 79) Beeri C. Rissanen J., On the decomposition of relational databases schemes, Workshop CERT-DERI, Toulouse, 1979.

- (BV 81) Beeri C. Vardi M., Formal systems for tuple and equality generating dependencies, Hebrew Un. of Jerusalem, Tec. Rep., 1981, to appear in SIAM J. Computing.
- (BB 83) Biskup J. Bruggemann H., Universal relation views : a pragmatic approach, Forschungsbericht, Nr 150, Abteilung Informatik, Universität Dortmund, 1983.
- (CFP 84) Casanova M. Fagin R. Papadimitriou C., Inclusion dependencies and their interaction with functional dependencies, J. Computer and System Science, 28.1, 1984.
- (CO 70) Codd E., A relational model of data for large shared data banks, CACM, 13.6, 1970.
- (Da 81) Date C., Referential integrity, Proc. 7th VLDB, 1981.
- (De 73) Delobel C., Contributions théoriques à la conception d'un système d'information, Thèse d'Etat, University of Grenoble, 1973.
- (De 78) Delobel C., Normalization and hierarchical dependencies in the relational data model, ACM TODS, 3.3, 1978.
- (Dem 82) Demolombe R., Utilisation du calcul des prédictats comme langage d'interrogation des bases de données, Thèse d'Etat, University of Toulouse, 1982.
- (FMU 82) Fagin R. Mendelzon A. Ullman J., A simplified universal relation assumption and its properties, ACM TODS, 7.3, 1982.
- (GMN 81 84) Gallaire H. Minker J. Nicolas J.M., Advances in database theory, Vol. 1, 1981, Vol. 2, 1984, Plenum Press, New York.
- (GMN 84) Gallaire H. Minker H. Nicolas J.M., Logic and databases: a deductive approach, Theoretical issues in databases, Benodet, France, 1984, to appear in ACM Surveys.
- (HLY 80) Honeyman P. Ladner R. Yannakakis M., Testing the universal instance assumption, Inf. Processing Letters, 10.1, 1980.
- (IL 82) Imielinski T. Lipski W., A systematic approach to relational database theory, Proc. ACM SIGMOD, Orlando, 1982.
- (Ko 81) Kowalski R., A metalanguage representation of relational databases for deductive question answering systems, Proc. 7th IJCAI Conf., Vancouver, 1981.
- (KU 80) Korth H. Ullman J., System U : a database system based on the universal relation assumption, XPl Workshop on relational database theory, 1980.

- (KYT 79) Kambayashi Y. Yajima S. Tanaka K., Problems of relational database design, Lecture Notes in Computer Science 132, Springer Verlag, Data Base Design Techniques I.
- (KTT 83) Kambayashi Y. Tanaka K. Takeda K., Synthesis of unnormalized relations incorporating more meaning, Information Sciences, 29, 1983.
- (Li 81) LIN S., Existential dependencies in relational databases, Ph D. Thesis, UCLA, 1981.
- (MMSU 80) Maier D. Mendelzon A. Sadri F. Ullman J., Adequacy of decompositions of relational data base, Proc. 20th IEEE Symp. on foundations of Computer Science, J. Computer System Science, 21.3, 1980.
- (MRSSW 82) Maier D. Rozenshtein D. Salvater S. Stein J. Warren D., Toward logical data independance : a relational query language without relations, Proc. ACM SIGMOD, 1982.
- (MU 80) Maier D. Ullman J., Maximal objects and the semantics of universal relation assumption, Report CS 800016 Suny at Stony Brook, 1980.
- (MUV) Maier D. Ullman J. Vardi Y., On the foundations of the universal relation model, previous version of this paper appeared as "The revenge of J.D.", Proc. ACM Symp. on Principles of Databases Systems, 1983.
- (Ni 76) Nicolas J.M., Mutual dependencies and some results on undecomposable relations, Proc. VLDB, Berlin, 1976.
- (PP 80) Parker D. Parsaye-Ghommi K., Inferences involving embedded multivalued dependencies and transitive dependencies, Proc. ACM SIGMOD, Los-Angeles, 1980.
- (Re 81) Reiter R., Towards a logical reconstruction of relational database theory, On Conceptual Modelling, Springer Verlag, 1981.
- (Sa 81) Sagiv Y., Can we use the universal instance assumption? Proc. ACM SIGMOD, 1981.
- (SDPF 81) Sagiv Y. Delobel C. Parker D. Fagin R., An equivalence between relational database dependencies and a fragment of propositional logic, JACM, 28.3, 1981.
- (Sc 80) Sciore E., The universal instance and database design, Tech. Rep. 271, Princeton University, 1980.
- (TKY 79) Tanaka K. Kambayashi Y. YAJIMA S., Properties of embedded multivalued dependencies in relational database, The transactions of IECE of Japon, E62.6, 1979.

- (Ve 83) Verroust A., Les schémas bien formés et opérations de mise à jour, Thèse 3ème Cycle, University of Orsay, 1983.
- (Th 84) Thalheim B., A complete axiomatization for join dependencies in relations, Technical Report, University of Dresden, 1984.
- (Za 77) Zaniolo C., Relational views in a database system support for queries, IEEE Comp. Soft. and Applications Conf., Chicago, 1977.

PORABLE COMPUTERS - PORTABLE OPERATING SYSTEMS

D. Wiegandt

CERN, Geneva, Switzerland

1. Portability

Portable: Capable of being carried by hand or on the person; capable of being moved from place to place; easily carried or conveyed.

The Oxford English Dictionary

1.1 Portable hardware

Hardware development has made rapid progress over the past decade. Computers used to have attributes like "general purpose" or "universal", nowadays they are labelled "personal" and "portable". Recently, a major manufacturing company started marketing a portable version of their personal computer. But even for these small computers the old truth still holds that the biggest disadvantage of a computer is that it must be programmed, hardware by itself does not make a computer.

"Software stands between the user and the machine."¹

For every major innovation a manufacturer of microcomputers is faced with the problem to provide software that can be used with his new hardware. To produce an operating system from scratch is a costly task often involving large teams of software engineers. Manufacturers therefore prefer to have existing software adapted to their specific product, "ported" to the new hardware.

1.2 Portable software

A piece of software is usually called portable if the effort to transport it is much lower than the effort to rewrite it for a new environment. There are several possible approaches to transporting software [1] :

1. Write a program that emulates an existing computer on the new hardware and then simply run the application programs or even the operating

¹ Harlan D. Mills

system of the older machine on the new hardware. This approach was taken by IBM when they introduced the Series /360 by providing emulators for the 7090 and the 1401, and in more recent times by a company called Dynamic Microprocessor Associates who offer EM 80/86, an 8080 emulator to run under CP/M-86 on Intel 8086 microprocessors. In spite of its success, this method is usually hampered by a severe performance penalty.

2. Define an instruction set and an architecture of a virtual machine. Then, write an operating system and all the necessary programming tools like compilers, assemblers, loaders, etc. for this virtual machine. All that is left to be done now is to write target-specific emulators that emulate the virtual machine on the new hardware. This is the approach taken by manufacturers providing the UCSD² p-System, which is based on the UCSD implementation of PASCAL, but also supports BASIC and FORTRAN. All programs are compiled into "p-code", which is then executed by an emulator on the target microprocessor. The UCSD p-system is available on some 9 processors, including Z80, 8086, 68000 and LSI-11. There is still a performance penalty associated with this approach, but a clever choice of the intermediate "higher level" language helps substantially to reduce the overhead imposed by emulation.
3. Write the whole programming environment, i.e. language translators, operating system, utilities and so on in a higher level language, carefully isolating machine-dependent parts. Transporting this system to a new environment then amounts to
 - a. writing a code generator for the high level language translator to generate code for the new target,
 - b. rewriting the machine dependent parts of the operating system,
 - c. finally recompiling the whole system using the adapted compiler.

This description is, of course, oversimplified, but it describes the principle of porting UNIX to a new environment. More details of this process can be found in [1] [2] [3] .

2. Operating systems for portable computers

Q: What is an elephant? A: An elephant is a mouse with an operating system.

Folk saying

² University of California at San Diego

2.1 Elements of operating systems for 8-bit microcomputers

The above folk saying is unfortunately not at all helpful to understand the purpose of an operating system. Here is a better definition:

An operating system is an organized collection of programs and data that is specifically designed to manage the resources of a computer system and to facilitate the creation of computer programs and control their execution on that system [4] .

Operating systems usually take over when the computer is initialized, and the first manifestation of the operating system's running is some prompt appearing on the screen. This prompt usually signals that the system is waiting for the user's commands. Here we have identified one principal task of every operating system: the interpretation and execution of the user's commands.

Usually a well defined module of the operating system is assigned to this task. Let us call this module CCP, Console Command Processor. To display the prompt on the screen and to read the command from the keyboard, CCP uses the services of another important component of the operating system, a module handling basic input/output operations. Let us call this module BIOS, for Basic Input/Output System. A microcomputer usually has some secondary storage, in general floppy disk units. Handling of disk accesses and file management are delegated to a third module, called BDOS for Basic Disk Operating System.

These three modules, CCP, BIOS, and BDOS are the essential components of a very popular microcomputer operating systems, CP/M (Control Program for Microprocessors), which was written in 1973 by Gary Kildall for the Intel 8080 microprocessor. As we shall see, control program is a better name than operating system for CP/M.

CP/M fans even call it a portable operating system, because "all you need to do to adapt CP/M to new hardware is to adapt the BIOS module to the new environment". This statement is obviously only true when you continue to run on a processor that executes 8080 machine instructions, e.g. a Z80. So the portability of CP/M is comparable to the one of MVS, IBM's most important operating system, which also runs on SIEMENS, Fujitsu, Amdahl, etc.

Looking at what we can do with CP/M, we soon realize that it is a single user single task system, which may just about satisfy the needs of a user of an 8-bit microcomputer, but it will certainly fail to cope with the requirements of a user of a 16-bit micro.

Before we go on to 16-bit systems, let us summarize, which functions of an operating system we have identified so far:

1. Command interpretation
2. Input/output handling

3. Implementation of a file system

4. System services

By the term "system services" we shall understand the provision of services of more general nature by the operating system, for example the conversion of the contents of a counter, driven by a quartz oscillator or by the line frequency, into a printable ASCII string containing date and time.

2.2 Elements of operating systems for 16-bit microcomputers

So far, we have dealt with 8-bit microcomputers. If we switch our attention now to 16-bit processors, we may ask the question what new requirements for an operating system arise from this hardware evolution. The first obvious improvement is the increased addressing range of 16-bit processors. With 8-bit micros, operating system and user program could not occupy more than 64k bytes of memory. This limit has been pushed to 1Mbyte for the most restricted 16-bit chip, the Intel 8086, other chips have an even bigger addressing range. So the operating system must provide means to enable a user to efficiently use more memory, i.e. memory management facilities.

The processing power of 16-bit chips has also increased significantly compared to 8-bit chips. To make efficient use of this additional processing power, the operating system should at least provide multiprogramming capability, or even allow multiuser operation. Most of the currently popular 16-bit operating systems do not even provide true multiprogramming, MS/DOS, for instance, only allows a printer spooler to run in the background concurrently with a single user program.

A third important area is the handling of sophisticated peripheral equipment. While 8-bit micros usually have floppy disk equipment attached that allows secondary storage amounts of a few megabyte, modern 16-bit computers are delivered with Winchester disk drives with capacities of a few hundred megabytes, and the number of workstations equipped with a high resolution bitmapped display unit is increasing continuously. This evolution has to be accompanied by a corresponding evolution of the I/O handling facilities of the operating systems.

Let us summarize the additional requirements of more advanced hardware:

5. Memory management

6. Process management

7. Improved file systems and I/O facilities.

We shall now try to deepen our understanding of operating systems by studying a very interesting system that is getting more and more popular in the area of microcomputers: UNIX.

3. A portable operating system: UNIX

3.1 History

Whoever attempts to tell the history of the UNIX system describes the situation in 1969 at Bell Laboratories that gave birth to UNIX: Scientists were frustrated by the failure of the Multics project, a collaboration between Bell Laboratories, General Electric and The Massachusetts Institute of Technology, from which Bell Laboratories had withdrawn. The computing facility at their disposal was a time-shared central computer, probably with rather bad turnaround times even for small programs.

One of these programs was "space travel", a program simulating the movements of the planets in the solar system, whose authors, Ken Thompson and Dennis Ritchie decided to use an obsolete PDP-7 with a CRT display unit to run this program interactively. Unfortunately the PDP-7 provided only an assembler and a loader, so an operating system, originally a single user system, was written by Ken Thompson. The start as a single user system may also be an explanation for its name, UNIX, as opposed to Multics, which was meant to become the state-of-the-art multiuser system.

The history of UNIX is tightly related to the history of the language C. UNIX was originally written in PDP-7 assembler, then rewritten, again in assembler, for a PDP-11/20 during 1971/1972. Thompson had been working on the language B, a descendant of BCPL. Attempts to use this language to rewrite the system failed, even after the introduction of variable types in the previously typeless language. After Dennis Ritchie had added structures and global variables in 1973, the new language became known as C, and a rewrite of the system in C was successful. The first publicly available version of UNIX, the Sixth Edition, was released in 1975.

Since then, UNIX evolved in different flavours on different hardware. Two major streams can now be distinguished:

1. AT&T UNIX System V, based on Bell Laboratories Seventh Edition UNIX is available for DEC Hardware and several microprocessors, e.g. Motorola 68000.
2. Berkeley 4.2bsd UNIX, based on a former VAX version of Bell Laboratories, has been enhanced with respect to that version by the implementation of demand paging and a number of useful features and utility programs. The ULTRIX system announced recently by DEC for VAXES is based on Berkeley 4.2bsd.

For more details on the history of UNIX see for instance [5] .

3.2 The UNIX file system

The UNIX file system was the first component of the system in the development process [5]. It did not yet have exactly the same structure as today, but one principle was already firmly established:

A file is a sequence of zero or more bytes. All additional structuring information has to be imposed and recognized by application programs.

Three categories of files are known by the UNIX file system:

1. Ordinary files
2. Directories
3. Special files

3.2.1 Ordinary files

As we have seen from the above-mentioned principle, an ordinary file is a string of bytes. An application program reading a text file will of course recognize a line structure, the convention being that all lines in a text file end with a newline character. The assembler generates, and the loader expects object files of a specific format. But in contrast to most other file systems this structure is entirely managed by the application program and not imposed by the operating system. There are no such things as records, blocks, blocking factors, blank padding and so on. Physical structuring imposed by the hardware (e.g. disk sector sizes) is completely hidden from the user.

3.2.2 Directories

A directory is the place where the link between a file name and a physical file on disk is kept. Each user has at least one directory of his own files, his "home directory". He may create any number of subdirectories to group related files together in a convenient way.

A directory is in principle an ordinary file containing directory entries and can be read like any other file, appropriate permission assumed. Modification, however, is only possible via system calls. In this way the system controls the contents of directories.

The structure of a UNIX file system is a rooted tree, the nodes being directories, the leaves being ordinary or special files. In standard UNIX (as opposed to distributed UNIX) each directory must appear as an entry in exactly one other directory, which is called its parent directory. There is one exception to this rule: the root directory (/), which is its own parent.

Each directory has at least two entries, the entry "." that points to the directory itself and the entry ".." that points to its parent directory.

When a user is logged on to a UNIX system, he is always associated with one directory or another, the working directory. The working directory is an attribute of every UNIX process. When you first log on, your home directory is your working directory, but provided you have the necessary access privileges you may change your working directory to any place in the hierarchy.

Files have names of 14 or less characters (Berkeley UNIX permits up to 255 characters in file names). Files may be specified to the system in the form of a path name. A path name is a sequence of directory names separated by slashes (/) ending in a file name. If the very first character of the path name is a slash, the search starts in the root directory (full path name), if not, the search starts in the working directory.

It is possible for an ordinary file to have more than one link, i.e. to be known under different names. These links may even be in different directories. All links to a physical file have the same status, i.e. the existence of a file is not bound to a specific directory. The file disappears only when the last link to it has been removed.

3.2.3 Special files

Each supported peripheral device has at least one special file associated with it. Special files may be read and written just like ordinary files, but here, the transfer requests result in an activation of the associated peripheral device. Special files are entries in the directory /dev. So if you want to write to magnetic tape for instance, you write to /dev/mt. Special files exist for main memory as well as for all peripherals connected to the system - disk units (partitions), terminal lines, ... Special precautions are of course taken to prevent illegal access to main memory and disk units.

The application of standard file operations on peripheral devices has the following important advantages:

1. Data transfer between a program and an ordinary file and data transfer between a program and a peripheral device is as similar as possible.
2. File names and devices names have identical syntax and meaning. A program that expects a file name as an argument can be given a device name.
3. Peripheral devices are subject to the same protection mechanisms (see below) as ordinary files.

3.3 Nonresident file systems

UNIX allows mounting of nonresident file systems. The root file system, however, has to be resident, as it contains the boot image. The mount function takes two arguments:

1. the name of an existing empty directory
2. the name of a special file which is associated with a storage medium that contains an independent file system with its own directory hierarchy.

"mount" substitutes the empty directory with the root directory of the mounted file system. There is no distinction between files on any mounted file system and files on a resident file system. The only restriction that applies is that links cannot be established between one file system hierarchy and another. This restriction has been introduced to avoid the overhead otherwise incurred with the removal of all such links when a file system is dismounted. Berkeley UNIX circumvents this difficulty by the introduction of "symbolic" links, a kind of aliasing of file names by indirect access.

3.4 Access control

Each UNIX user has a unique identification, consisting of a user number and a group number. These numbers were assigned to him by the system administrator, when the user registered. When a user logs in the login program will set the numbers associated with this user after having read the corresponding entry in the password file.

When a user creates a new file, this file is marked with the user number and the group number of its owner. Associated with all files (ordinary files, directories and special files) is a set of nine protection bits that specify

1. read permission, write permission and execute permission for the file owner
2. read permission, write permission and execute permission for other members of the owner's group
3. read permission, write permission and execute permission for all other users.

Another bit is only used with executable files, it is called the set-user-ID bit.³ Normally, a program is executed with the user identification of its caller. If the set-user-ID bit is on, however, the program is executed with the identification of its owner. This mechanism is by the way protected by a patent

³ There is also a set-group-ID bit.

held by Dennis Ritchie.

A good example for the use of the set-user-ID bit is the program `passwd`, a program that anybody can use to introduce or modify his own password. Clearly this program needs write access to the system file that holds all information on registered users, but this file in turn must certainly be protected against unwanted modification. The strategy followed here is the following:

1. Create the `passwd` file as owned by user root, a user that is necessarily present in all UNIX systems and has user-ID 0.
2. Give it read access by everybody, because the encrypted passwords stored in the file cannot be decrypted anyway.
3. Give it write access only by its owner, i.e. root.
4. Create the `passwd` program as a file owned by root and executable by everybody.
5. Set the set-user-ID bit for this program, so whenever this program is executed it will pretend to be executed by root.

In this way you can assure that write access to the `passwd` file is limited. The identity of the actual caller of the program is also always available, so the called set-user-ID program can always verify access rights and take appropriate measures if necessary.

One particular user ID is exempt from file access constraints: the super-user. The system manager uses this feature to avoid interference from the protection mechanism for tasks like file backup and restore. Data that must not be read even by the super-user will have to be encrypted. There are system utilities to do that.

3.5 Input/output primitives

The lowest level of I/O is done by system calls. All Input/Output in UNIX is done to or from files, so the same system calls are used to do I/O to disk files, terminals, any other peripheral device. Some of the system calls may not be applicable to all devices, however, e.g. it does not make sense to do positioning on terminal input or to read from a line printer. I/O system calls are functions that return integer values.

To read or write an existing file, you have to open the file by

```
filep = open(name,flag);
```

where name indicates a path name (see above), flag indicates whether the file is to be read or written, and the result of a successful operation is a small integer value called a file descriptor, which is to be used to identify the file in subsequent I/O system calls related to the same file.

It is an error to open a file that does not exist. To open a new file or to overwrite an existing one, you use the creat system call:

```
filep = creat(name,perms);
```

where perms is the protection to be associated with the newly created file. Recent versions of UNIX provide a "Create if not existent" flag for the open system call, thus combining open and creat.

Reading and writing of files is normally sequential, there is, however, a way to read or write in arbitrary order by positioning in the file before transferring data (see below). UNIX maintains a pointer for each open file to the next byte to be read or written. If n bytes are transferred, this pointer advances by n bytes. Reading or writing is done by the following functions:

```
n = read(filep, buffer, count);
n = write(filep, buffer, count);
```

A maximum of count bytes is transferred between the file specified by filep and the array specified by buffer. The result n is the number of bytes actually transferred. For write operations this should always be equal to count, except in error cases, e.g. end of tape. The result n of a read operation can be less than count if only n bytes remain before the end of the file, n = 0 indicates that the end of the file has been reached.

There is a system function that permits to move the position pointer of a file to the desired location in the file:

```
loc = lseek (filep, offset, base);
```

The pointer associated with the file specified by filep is moved by offset bytes relative to base. Base can be the beginning of the file, the current position in the file, or the end of the file. The resulting offset from the beginning of the file is returned in loc. lseek can thus be used to determine the current position by specifying an offset of zero relative to the current position.

Another system function is the function

```
res = ioctl(filep, request, arg);
```

which is primarily used to control operating characteristics of peripheral devices, e.g. baud rates of terminal lines, echoing etc.

The function close serves to terminate the treatment of a file, unlink removes the entry of a file in a directory. Note that the file is deleted only after the last link to it has been removed.

UNIX does not have a mechanism for file locking at the user level, so any number of users can simultaneously read and write (!) a file. There are, however, internal interlocks to maintain the file system's consistency in cases where files are manipulated by more than one user [6] .

3.6 Implementation of the file system

3.6.1 Structure of a file system

The following description of the UNIX file system applies to Version 7 UNIX. The availability of disk units with capacities of several hundred megabytes has led to modifications of the file system implementation to improve the efficiency and the security of the file system. The file system implemented in Berkeley 4.2bsd for instance was heavily modified compared to Version 7. For details of these modifications see [7] .

The disk space of a UNIX file system is split in 4 segments:

1. Block zero which may be a boot block, is unused by the file system.
2. Block one, the so called super block, which contains the size of the file system and the boundaries of the other regions.
3. The i-list (index list), a sequence of i-nodes (index nodes), the number of which determines the maximum number of files in this file system.
4. Storage blocks

Each of the i-nodes in the i-list is a structure that may contain a file definition. The index of an i-node in the i-list, called the i-number, plus major and minor device number (see below) uniquely identify a file. A directory entry consists of an i-number and a file name.

Each i-node in turn holds the following information about a file:

1. user and group ID of its owner;
2. the protection bits;
3. the address of 13 physical blocks of the file;
4. its size in bytes;
5. time of creation, last use, and last modification, in GMT seconds;
6. the number of links to the file (how many times it appears in directories);
7. the kind of file it is (ordinary, directory or special file).

3.6.2 Access to blocks of a file

The addresses of the physical blocks of the file held in the i-node have to be interpreted in the following way (UNIX Version 7):

- The first ten addresses point to the first ten data blocks of the file. If a file is bigger than 5120 bytes, the eleventh address points to a block that contains up to 128 addresses of further data blocks.
- Even bigger files (> 70656 bytes) require double indirection via the twelfth address (≤ 8459264 bytes) or even triple indirection via address 13.

This mechanism allows files of up to 1 082 201 088 bytes. Very large files thus require up to four disk accesses to get to a single data block. This overhead is eliminated in practice by a cache mechanism (see below).

To reduce the levels of indirection for files of sizes up to 2^{32} bytes, the Berkeley 4.2bsd file system uses block sizes of 4096 bytes or greater [7].

Only blocks that have been written into are actually allocated. There is a zero address for blocks that have never been used in the i-node structure, and null bytes are returned when you attempt to read such a block. In this way UNIX supports sparsely written random access files.

3.6.3 Creation and deletion of files

Files are created by the allocation of an i-node and the construction of a directory entry that contains the file name and the number of the i-node just obtained.

Making a link to an existing file amounts to creating a directory entry containing the new file name and a copy of the i-number of the existing file.

Files are deleted by decrementing the link count in the i-node and removing the directory entry. If the link count reaches zero, all disk blocks of the file are freed and the i-node is deallocated.

Free space in the Version 7 file system is maintained by a linked list of available disk blocks. Every block in the chain contains a pointer to the next such block and up to 50 addresses of free disk blocks. A single read operation is thus sufficient to obtain 50 free blocks and a pointer indicating where to find more space. Berkeley 4.2bsd UNIX introduces a further subdivision of a file system into cylinder groups to increase locality of access with today's Gigabyte disks. The information on free blocks in a cylinder group is kept in a bitmap.

3.6.4 Special files

The above description of i-nodes holds for ordinary files and directories. For a special file, only the first address in the i-node is used and this specifies

an internal device name consisting of a major and a minor device number. The major device number is used as an index into a table holding driver entry points, the minor device number is handed over to the driver and selects, for example, a particular terminal port or a particular disk partition.

3.6.5 Path name interpretation

During the execution of an open system call, the i-node corresponding to the path name given as an argument to open has to be found. Depending on whether the path name starts with a slash, the search either starts at the root i-node, for which device number and i-number (always 2) are known , or in the working directory, whose device number and i-number are also known as a result of the last chdir (change directory) call.

For every component of the path name, the corresponding i-node is found. If it is the last component, we have reached the final i-node, else it must be a directory, otherwise the path name is illegal.

There is one special case that has to be taken care of in this search: a file belonging to a mounted file system. Each intermediate i-node is therefore checked if it is a "mounted-on" i-node. If so, the mount table is accessed that holds a correspondence between a device number i-number pair and the number of the device on which the mounted file system resides. This number replaces the current device number and the i-number of the root directory (2) replaces the current i-number. From then on the search continues in the mounted file system's hierarchy.

3.7 Block and character I/O

The I/O system of UNIX is split into two parts: the block I/O system and the character I/O system. Thompson points out in [8] that these parts should rather have been called "structured I/O" and "unstructured I/O". In addition to the minor and major device number, each device is characterized by a class: block or character.

For each class there is an array of entry points of device drivers (configuration table), which is indexed by the major device number and the function to perform, e.g. read, write, open, close. These arrays of entry points are the only connections between the system code and the device drivers. It has turned out that this clean separation facilitates the creation of new device drivers enormously.

3.7.1 Buffering with Block I/O devices

Reading and writing appear to be synchronous and unbuffered to the UNIX user. But there is a fairly complex buffering mechanism provided by the system that greatly reduces the number of physical I/O-operations. A physical transfer

is only necessary as a result of a read or write operation, when the corresponding data block is not yet in the internal buffer cache of the system. Otherwise, the required number of bytes are copied to or from the data buffer of the program, and in case of a write operation, the system buffer is marked to be written out.

The system also recognizes when a program treats a file sequentially and asynchronously pre-reads the next block. In this way the running time of most programs is significantly reduced at the expense of little system overhead.

There are, however, some problems associated with this cache mechanism:

1. Error reporting and error handling are made quite difficult by the asynchronous nature of the physical I/O operation.
2. In case of a system crash it is very likely that there are I/O operations that are logically, but not physically complete, because the corresponding system cache blocks have not yet been written out. Use of a system primitive that periodically flushes all outstanding I/O activity does not completely solve this problem.
3. The physical I/O sequence does not necessarily correspond to the logical I/O sequence. This could have disastrous effects for writes on sequential devices like magnetic tapes, thus the number of outstanding writes to tape units has to be limited to one.

3.7.2 Terminal I/O as an example of character device I/O

For the handling of character-oriented devices the system provides character lists. A character list is a queue of characters, that may be filled by one program and emptied by another. There are system routines for the handling of these character lists that will take care of automatic storage allocation and deallocation, as the need arises.

Character output from a program to a device is implemented by passing characters from the user program to the appropriate output queue until a maximum is reached, at which point the user program is put to sleep (see below). Physical I/O is started as soon as there is something on the queue and is sustained by hardware transfer completion interrupts. When the number of characters falls below a certain intermediate level, the user program is woken up and may continue to fill the queue up to the maximum value.

For a terminal there is some additional code that provides optional special actions on certain output characters, e.g. delay introduction after carriage return or form feed characters, conversion of tabulation characters to an appropriate number of spaces etc.

A terminal has two character input queues, called raw queue and canonical queue. Characters typed on the terminal are put on the raw queue.

If the terminal is in its standard operating mode (cooked mode), input editing, e.g. treatment of backspaces and some handling of special characters will take place, e.g. the necessary actions on receipt of flow control characters xoff/xon, echoing of the input etc., but the program waiting for terminal input is not notified until the input line is complete. Upon receipt of a line termination character, the contents of the raw queue are copied to the canonical queue and the user program is woken up.

If the terminal is in raw mode, characters will be passed unmodified immediately to the program reading terminal input. This mode is used for special applications like full screen editors or special devices that use a serial interface like a terminal but require a particular treatment. Note that in raw mode no output processing is done either.

An intermediate mode is cbreak mode, where all input characters are immediately made available to the user program, but flow control and handling of interrupt characters are still enabled. Output processing is done as for cooked mode.

3.8 Processes

Ritchie and Thompson [9] define an "image" to be the current state of a pseudo-computer. This image comprises memory and register contents, open files, current directory and so on. A process is said to be the execution of an "image", which must then be in main memory. During the execution of another process, it may remain in main memory or be swapped out to secondary storage if necessary.⁴

The virtual memory of a process is divided into a user part and a system part. The user part can be further subdivided into:

1. a text segment which contains the program code, is write protected and is shared between all processes executing the same code;
2. a non-shared writable data segment;
3. a user stack.

There is also a small data area in system space associated with a process that contains data needed only when the process is active, like information on open files, stack area for the system phase of the process and so on. This area can be swapped out with the process.

⁴ Berkeley 4.2bsd UNIX implements a different kind of memory management, "demand paging" [10].

Finally, each process is represented by an entry in the process table, which is allocated at process creation and freed at process termination. Each process in the system has a unique process-ID associated with it.

3.8.1 Process creation

Processes are created with the system call fork. The only exception to this rule exists during the boot process, where some permanently running processes (swapper and init) are created "manually". The result of the execution of

```
process_id = fork();
```

is that the running process splits into two independently executing processes. Each process has its own copy of the original memory image, and all open files are shared. The only difference between these processes is that one is considered to be the parent process, the other one the child process. This is accomplished by fork returning twice, once to the parent process, furnishing a non-zero process_id, and once to the child process returning zero.

3.8.2 Execution of programs

A process may use the system call

```
exec( file, arg1, arg2,...,argn);
```

to read in and execute the program named by file. The string of arguments arg1 ... argn is passed to that program. All code and data of the calling process are replaced from the file, but open files, current directory and process relationships are not changed, nor is the process-ID. There is no return to the calling process from the exec system call except if the program to be executed could not be found or execute permission was not granted.

3.8.3 Pipes

Communication between related processes in UNIX can be achieved using "pipes". A pipe is a unidirectional communication channel. The system calls write and read are used for the communication. A child process will inherit pipes that a parent process has set up. Pipes are implemented as internal FIFO buffers. If a process reads a pipe, which is empty, it will wait, if a process writes into a pipe which is getting too full, it is also suspended, until there is room again in the pipe. If the process on the write side closes the pipe, the reading process will see the end of file condition.

This kind of interprocess communication is a very valuable tool which is mainly exploited by the shell, the UNIX command interpreter, for chaining commands together. It is, however, in its original form, limited to communication between processes that have a common ancestor. More recent versions of UNIX implement generally usable interprocess communication facilities [11].

3.8.4 Scheduling and swapping

User programs in UNIX are executed by a user process. When the program requires a system function to be executed, it calls the system much as if it were a subroutine. This call uses the trap mechanism of the hardware to switch to kernel mode, thus to a new environment. From then on the process is said to be a kernel process. User process and kernel process never execute simultaneously. The user phase and the kernel phase of the process each have their own stack. Note that interrupt handling also forces an interrupted user process into kernel mode. Once all kernel activity has finished, a process returns to user mode.

Processes frequently cannot proceed unless some condition is true, e.g. an I/O operation has finished, a child process has terminated etc. Such process then waits for a specific event to occur. Events in UNIX are represented by arbitrary integers. There is a convention to use addresses of tables associated with those events, for instance, the termination of a child process is associated with the process table entry of the parent process. When a child process terminates, it will signal this event with the parent's process table entry address as an argument. Signaling an event for which no process is waiting, has no effect, signaling an event for which many processes are waiting will wake all of them up. As there is no memory associated with an event, waiting on an event which has already happened results in eternal sleep. This is one of the features of UNIX that render its use for real-time purposes somewhat difficult, and allow its use in multi-processor configurations only after substantial modification.

At any time, only one process in UNIX is executing, all others have called wait on event for one reason or another. Events these processes were waiting for may have occurred in the meantime, so many of them are "runnable". Whenever the executing process relinquishes the cpu or an interruption of some kind occurs, the next process to execute is selected. The selection is done based on priorities. The priority of a process in kernel mode is determined by the code that called wait on event, waiting on disk I/O is associated with higher priority than waiting on terminal I/O and so on. Kernel processes always have higher priority than processes in user mode. Priorities of user processes are recalculated every second based on the ratio between cpu time to real time since the last check. The higher this ratio, the more the priority is reduced. Interactive processes tend to have a low cpu time to real time ratio, thus interactive response is maintained without special arrangements.

Let us summarize the scheduling policy of UNIX:

- Processes in kernel mode are picked first and processes in user mode second.
- Priorities are recalculated every second. Looping user processes, all other conditions being equal, will be scheduled round-robin with a one second time slice.
- A high priority process waking up will preempt a running low priority process.

- No process can use its priority to hog the cpu, because its priority will drop. No low priority process can be ignored for a long time, because its priority will rise.

Process switching in fact involves yet another component of the system: the swapper. The swapper process, process 0, is one of the permanently running processes generated at boot time. Whenever a process switch is initiated, control is first transferred to the swapper process. The swapper process inspects the process table to find a swapped-out process that is runnable. This process is then loaded into memory and competes with the other processes for the cpu. If there is not enough free memory available to swap the selected process in, the process table is searched for candidates to be swapped out to disk to free a sufficient amount of memory.

Two selection algorithms are used by the swapper:

- The selection criterion for swapping in is the secondary storage residence time of the process, its priority and its requirements for primary storage, with a slight penalty for big jobs.
- Candidates for being swapped out are primarily jobs waiting for a slow event (e.g. terminal I/O). Selection then depends on the time of residence in primary memory, big jobs also being treated less favourably.

This area again is a somewhat problematic area of UNIX. Small systems with little primary memory tend to do a lot of swapping, which by itself is not so bad, but can lead to bad bottleneck problems when the swapping device is also the main file storage device. Big systems often have enough primary memory to avoid total swapping and can afford to have a separate swapping disk.

Reasonable performance of a UNIX system can only be expected when certain minimal hardware requirements are fulfilled, e.g. > 1 Mbyte of main memory, reasonably fast disks (access times < 100 ms) etc. A paper dealing with these minimal requirements is [12] .

3.9 The shell

UNIX differs from most other operating systems by not having its command interpreter integrated in the operating system kernel. The UNIX command interpreter is called the shell, because it encloses the kernel like a shell encloses its interiors. It is an application program without special privileges and can easily be replaced by another command interpreter. There are nowadays different "shells" available on UNIX. All UNIX versions have the original shell, often called the "Bourne shell" after its author S. R. Bourne. Most UNIX versions have the c-shell, written at UC Berkeley by William N. Joy, which utilizes a command language resembling the language C, and has other useful features [13] .

The simplest form of a UNIX command line is a command name followed by arguments for that command:

```
command arg1 arg2 arg3
```

The components of a command line are separated by spaces. The shell will search a file with the name "command". "command" is the name of a program or an executable file. It can be any valid path name (see above). If the file is found, it is loaded and executed. The arguments specified in the command line are accessible to the program that is being executed. When the program terminates, the shell regains control and prompts for the next command.

If the program to be executed cannot be found in the current directory, the shell attempts to find it in a user-definable set of standard directories.

3.9.1 Standard I/O and redirection

All programs executed by the shell start with three open files:

1. Standard input, file descriptor 0
2. Standard output, file descriptor 1
3. Standard error output, file descriptor 2.

These files normally correspond to the user's terminal keyboard and terminal screen. All UNIX utilities use this convention.

It is, however, very easy to change the standard assignments of these file descriptors from the terminal to any other file. If an argument to the command is prefixed by a ">" character, output will, for the duration of the command, be directed to the file named after the ">".

```
ls >myfiles
```

will list all files in the working directory and write its results into a file called "myfiles". This file is either created or its previous contents are overwritten.

Similarly, if an argument is preceded by a "<", input to the program will be taken from the file named after the "<", instead of being read from the keyboard.

File descriptor 2, the standard error output, usually remains coupled to the terminal screen to prevent error messages from disappearing silently into an output file.

3.9.2 Filters

A unique feature of UNIX is the possibility to direct standard output from one command into standard input of another. A sequence of commands separated

by vertical bars "|" in a command line causes the shell to start all these commands simultaneously and to set up pipes (see above) in such a way that standard output of the program on the left of the "|" is linked by a pipe to standard input of the program on the right of the "|".

```
ls | pr -2 | lpr
```

ls lists the files in the current directory. Its results are handed to pr that formats its input onto pages with dated headings in two column format (the -2 argument). The output of pr in turn is passed to the printer spooler lpr.

Other operating systems require the use of temporary disk files to do a similar sequence of operations, e.g.

```
ls >temp1  
pr -2 <temp1 >temp2  
lpr <temp2  
rm temp1 temp2
```

where the last step, the removal of the intermediate files is only too often forgotten.

Programs like pr which manipulate data read from standard input and produce results on standard output are called filters. There are many filters available as UNIX utilities.

3.9.3 Running a program in the background

Programs that do not need supervision and do not read from the keyboard can be run in the background. When you terminate your command line with an ampersand (&) before the carriage return, the shell will start the execution of the command, but not wait for its termination, it will instead display the process identification number of the program running in the background and prompt you for a new command. Output produced by the program in the background will still appear on the terminal if you do not redirect it. Most versions of UNIX require a special command when you want the program in the background to continue even in case you log out.

3.9.4 File name generation

Metacharacters can be used in a command line to generate lists of file names. The shell will expand an argument containing metacharacters into a list of file names and pass that list to the program invoked. Valid meta-characters are:

- * matches zero or more characters in a file name, but not a leading period (.).
- ? matches any single character in the name of a file

[..] matches all characters within the brackets, ranges may be defined by placing a hyphen between two characters.

3.9.5 Command files

The shell itself is a command like any other, and can be called recursively. The shell reads commands from standard input and executes them. Commands that have been stored in a file can be executed by invoking the shell recursively and redirecting its standard input such that the shell reads this file:

```
sh <comfile
```

Comfile may for instance contain a series of commands to compile and run a program.

3.9.6 Other capabilities of the shell

The shell is both a command interpreter and a programming language interpreter. As a command interpreter, it executes commands that a user types in. As a programming language interpreter, it interprets statements stored in command files, commonly called shell scripts. The shell programming language comprises features like

- Variable assignment and substitution
- Control structures like if-then-else, case
- Repetitive command execution (for, while, until)
- Reading and prompting for user input
- Substitution of a command by the result of its execution

For more details please consult [5] .

3.9.7 Implementation of command interpreters

A command interpreter, normally a shell, is invoked when a user logs in successfully. This shell then sends a prompt to the screen and issues a read command to get a command line from the keyboard. When the line terminating character is entered, the shell will analyze the command line received. The arguments in the command line, if any, will be put in a form suitable for the exec system call. A child process is then created by the execution of the fork system call. This child process performs an exec system call to load and execute the desired command with the supplied arguments.

The parent process waits for the termination of its child. When this has happened, the parent process knows that the command has been executed, and branches back to prompting and reading command lines from the keyboard.

For the execution of commands in the background the parent process does not await the child's termination, but prints the result of the fork call and branches back to prompting and reading the next command.

Input/Output redirection is taken care of by the child process. Before it executes the command it makes the file descriptor 0 or 1 refer to the file named after the < or > sign. It does so by closing the file with the appropriate file descriptor and then opening the named file. By convention, the smallest file descriptor available will be returned by open, thus nicely establishing the desired connection.

Filters redirect standard I/O to pipes instead of ordinary files in a completely analogous way.

The shell normally stays in an eternal loop, attempting to read the next command. An end of file condition, however, will terminate the shell. This also covers the case of a shell reading a command file. It will terminate when the end of the command file is reached.

3.10 The UNIX programming environment

UNIX is different enough from most other operating systems that it is definitely justified to talk about a specific UNIX programming environment. There are several design concepts of UNIX that merit special attention:

1. Directories are normal files with the exception that they cannot be modified by an ordinary program. Modification of a directory entails modifications of file system information kept in i-nodes, a task which is better left to the kernel. Any ordinary program can, however, read and interpret the contents of a directory, so a directory listing program does not need any privileges.
2. A file is a sequence of bytes. At the first sight this may seem a deficiency of the system because the application program then has to worry about the structure of the data in the file, but it turns out to be a blessing. A user is not at all concerned with physical layouts of disks, things like sector size and track size are hidden from him. There is nothing like record headers or other system generated information in a file, every byte in a file has been put there by the user himself. There is no need to know the file size in advance, because space on the disk is allocated while the file is written. All files are identical in form, there are no "access methods" for different kinds of files.
3. All input/output operations deal with files. There are only a few standard i/o primitives that apply to all files, and peripheral devices are special files in UNIX (see above), which are treated in exactly the same way as disk files. A UNIX program reads data from a file and writes data to a file, and there is no need to care whether the file is actually a keyboard or a disk file.

4. Input/output can be redirected. The ease of input/output redirection together with the just mentioned equivalence of disk files and special files helps enormously in the debugging phases of a program, because one can easily run the program interactively, feeding data into the program from the keyboard and receiving its output on the screen. Once the program is considered sufficiently bugfree, I/O is redirected to disk files or pipes.
5. Tools and pipes allow problem solutions without writing application programs. UNIX comes with a rich set of small, generally useful utilities, like sort programs, pattern-matching programs etc. Quite frequently the combination of tools by pipes into a chain of small programs solves exactly the programmer's problem, relieving him from the burden of writing a special application program for a possibly single occurrence of a problem. Shell programming is also very valuable in this context.
6. There is also a reverse side of the coin. UNIX does not only have fans, but there are also people, who find it horrible [14]. One reason for this attitude is the fact that "learning UNIX" is not a matter of five minutes, the entrance threshold of the system is rather high. Commands often have cryptic names, and you can hardly use the system without a UNIX manual within reach. Another reason is that UNIX tends to be very terse, "no news is good news" seems to be the underlying philosophy. This can at least partly be explained by the fact that programs that are likely to be linked by pipes to other programs must not do things like prompt for input or even send error messages to the standard output file. This, by the way, is another good reason for leaving the standard error output file directed to the terminal screen. It cannot be denied either that UNIX was conceived at a time when hardcopy teletypes were the standard user terminal, and economy on key strokes was important. Recent versions of UNIX, however, got adapted to the world of CRT terminals, some are beginning to make use of the capabilities of modern bitmapped display terminals by provision of window management.

4. Conclusion

The preceding discussion of some UNIX internals and some UNIX features was meant to demonstrate that UNIX is without any doubt one of the most interesting systems in use for computers today. Its use for personal computers will continue to grow for a variety of reasons, limiting factors being its non-trivial demands on hardware and the fact that its multi-user capabilities may be a luxury rather than an asset for small machines. UNIX machines linked together in a local area network and running a "distributed" version of UNIX, e.g. the "Newcastle connection", could turn out to be a very interesting configuration of the future.

References

- [1] P.J. Jalics and T.S. Heines, "Transporting a Portable Operating System: UNIX to an IBM Minicomputer," Comm. ACM Vol. No. 12 p. 1066 (December 1983).
- [2] M. Tilson, "Moving UNIX to New Machines," BYTE Vol. 8 No. 10 p. 266 (October 1983).
- [3] Nai-Ting Hsu, G. Skinner, and J. Zelitzky, "How to Port UNIX to a New Microprocessor," COMPUTER DESIGN Vol. 23 No. 6 p. 173 (June 1984).
- [4] H. Katzan jr., Operating Systems, Van Nostrand (1973).
- [5] S.R. Bourne, The UNIX System, Addison Wesley (1982).
- [6] D.M. Ritchie, "The UNIX I/O System," UNIX Programmer's Manual Vol. 2B(January 1979).
- [7] M.K. McKusick, W.N. Joy, S.J. Leffler, and R.S. Fabry, "A Fast File System for UNIX," UNIX Programmer's Manual Vol. 2C(August 1983).
- [8] K. Thompson, "UNIX Implementation," Bell System Technical Journal Vol. 57 No. 6 p. 1931 (July 1978).
- [9] D.M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," Bell System Technical Journal Vol. 57 No. 6 p. 1905 (July 1978).
- [10] O. Babaoglu and W. Joy, "Converting a Swap-Based System to Paging in an Architecture Lacking Page-Referenced Bits," Operating System Review Vol. 15 No. 5 p. 78 (December 1981).
- [11] W. Joy, E. Cooper, R. Fabry, S. Leffler, K. McKusick, and D. Mosher, "4.2BSD System Manual," UNIX Programmer's Manual Vol. 2C(August 1983).
- [12] R.C. Gammill, "The UNIX Operating System," SIGSMALL NEWSLETTER Vol. 7 No. 3 & 4 p. 31 (December 1981).
- [13] W.N. Joy, "Introduction to the C shell," UNIX Programmer's Manual Vol. 2C(August 1983).
- [14] D.A. Norman, "The Trouble with UNIX," DATAMATION Vol. 27 No. 12 p. 139 (November 1981).

SINGLE USER SYSTEMS

Ian Willers

CERN, Geneva, Switzerland

Abstract: The first part of the talk will be devoted to establishing concepts and trends in interactive computing technology and is intended to provide a framework. I will then discuss personal computing architectures of today and planned architectures of the 1990s. I will present current personal computer environments for the programmer and for the user. Scenarios for future computing environments will be developed. Finally, I will open a discussion on the social implications of personal computers. Many of the ideas in this paper are covered to greater detail in the proceedings of the Capri conference held this year [1].

1. Introduction

The computing world has been dominated during the last two decades by the multi-user, time-shared machine. The reason for this was an economical one of sharing expensive items between a large number of users. Since many of the components are no longer expensive we have seen the introduction of single user systems in the form of personal computers and personal workstations.

Network technology has evolved so that single user systems may access server machines which offer services which are to be shared amongst the users. The servers will give access to expensive devices which are normally attached to multi-user, time-shared machines. Economics is still the driving force behind the design of such systems.

What is it that makes the single user system so interesting? The dedication of a single processor with a fast graphics interface to a large bit map display has given users a new level of user interaction. Due to the high speed of the graphics the user interface is no longer restricted to being verbal but can include images that can be moved on the screen by the user or by the computer to give animation. Graphical applications now run consistently at high speed no longer being swapped out as another user's program runs.

Adding a new server to a system involves interfacing a device to the network rather than incorporating it into an already complex system making both installation and maintenance an easier task.

At CERN we have begun the introduction of Apollo personal workstations and we have tested a number of other types. The highest priority has been to integrate the Apollo with networks at CERN. The work on the machine itself is based on Graphics and Fortran development. We have developed 2-D graphics using GKS, a 3-D graphics system, PIONS, and a graphics editor, GUI. The kernal of the CERN Fortran library has been ported.

2. Evolution of Technology

If we look at the second half of this century, it began with expensive single user systems allocated to privileged individuals. After three decades of large multi-user, time-sharing systems we can see a trend back to single user systems. Only now more or less any individual can afford to have his own personal computer and professionals are purchasing personal workstations which retain the advantages associated with multi-user, time-sharing systems!

Let us now look at developments during the period 1950-1999.

1950's stand alone, single user systems: This often required booking schemes with computers in operation 24 hours each day. By the end of the decade batch services were in operation. A computer ran a job to completion before taking the next job from the input device. Programs were numerical, data was transient and the first mnemonic assemblers and compilers were conceived.

1960's multi-user, time-sharing machines: Computers were still very expensive and the emphasis was on finding efficient ways of exploiting the hardware. Time-sharing systems such as Multics [2] were developed. The complexity of the systems grew as they were used to manage banks and airline reservation systems. Complex databases were constructed.

1970's big mainframes and software engineering: New machines were developed to handle larger and larger problems. Building software to handle such problems often with large teams of programmers became analogous to a large engineering project. This led to new programming methodologies, abstraction mechanisms and the life cycle concept of software.

1980's component technology. Both hardware and software experts are now concentrating on constructing systems from modular components. Single user systems are linked via networks to each other and to services normally supplied by a time-sharing system. Personal computers have hard interfaces enabling users to tailor their systems. Languages such as ADA [3] and Modula-2 [4] support modules and check the software interfaces between modules. There is also an enormous development of user interfaces using powerful graphics systems on single user machines.

1990's knowledge engineering: Future computers will support expert systems. All types of disciplines will be utilising computers giving them a knowledge support environment.

3. Personal Workstation Genealogy

The early work on personal workstations began in the 1970's at the Xerox Palo Alto Research Center, commonly called Xerox PARC. Their dream was a machine which was the size of a book and resembled in very many ways a sketch-pad. The result of this was a personal workstation. The Alto [5] was begun in 1973 and included a bit-mapped screen and a graphical input device. Xerox has since constructed two research machines, which are called the Dolphin and the Dorado, and an office workstation which is called the Xerox Star.

The Massachusetts Institute of Technology, MIT, developed Lisp [6] machines called CONS and CADR. From these a commercial machine, the Symbolics 3600, was manufactured primarily to run the large algebra program MACSYMA [7].

In Europe, ETH in Zurich produced the Lillith [8], with a microcode tailored to the language Modula-2, and Inria in Paris has an office project using the Buroviseur [9].

The American computer manufacturers have now produced a number of personal workstations, the SUN, Lisa, Corvus, PERQ, Apollo etc.

4. Recent Events

In order to get necessary performance the control processor of a personal workstation was constructed out of bit slices which were microprogrammed. In 1979 the Motorola 68000 microprocessor was incorporated instead. The Motorola 68000 is a 16 bit microprocessor which allows 32 bit operations and a performance of 1 MIPS.

As personal workstations form one component of a system, the system was connected using networks. In 1980, Ethernet [10] became commercially available so that personal workstations now either connect directly to Ethernet or will gateway from its own network to Ethernet. Ethernet was another Xerox product.

In 1982, the Motorola 68010 microprocessor was introduced giving a virtual memory of 16 Megabytes to the user. This gives the personal workstation the power to run programs which previously could only run on mainframes. It also gave Motorola 68010 based machines an immediate advantage over bit-sliced machines where microcoding of virtual memory has proved to be difficult.

This year has seen the introduction of very high quality bit-map displays both in black and white and in colour. The improvement in quality has kept within the present price structure.

5. The Personal Workstation as Part of a System

Since the personal workstation is a component of a networking system the operating system should be prepared to aid the user in accessing different components of the systems. A number of solutions to this problem have been proposed and are outlined below.

1. Enhance the operating system by adding remote login and file transfer. This is the solution adopted by UNIX¹ [11] and UNIX-like systems which support the TCP/IP protocols [12]. The personal workstation remains isolated but files can be transferred easily from one system to another using the FTP program. The user can investigate the state of or use another computer by logging in using TELNET program. Computers attached to the network are recognized by their network address.
2. The kernel of the operating system can be written to use message passing whenever the service being requested could be on another computer. The other computer must run a kernel that understands these messages. However the message may be passed locally or via the network. The address space of objects has to be extended so that each object in the network can be addressed. In the case of Apollo the messages are passed over the Domain [13] token ring network and each object has a unique 96 bit address. Programs and files are paged across the network.
3. The third approach is that of the Newcastle Connection [14] as applied to UNIX and UNIX-like systems. In this case calls to the operating system kernel are intercepted and the code determines whether the call is local or not. In the case of a non-local call, a message is passed to the other computer using a remote procedure call protocol. The results are then returned to the caller via another message. Local calls are passed on to the local kernel giving some inefficiency. The address space of objects in the network is extended by using device files referring to other computers.
4. Terminal emulation is the slowest and least powerful technique of integrating system components. It is however the most used method of communicating between single user systems and other computers. The terminal emulator mimics the operation of a terminal to the

¹ UNIX is a trademark of Bell Laboratories

computer. It is capable therefore of remote login and a slow form of file transfer. By listing a file, the file's contents are passed over the terminal line and are copied to the disk of the single user system. This transfer is usually limited to the speed of the terminal line which is normally between 300 and 9,600 bits per second. This technique is the most commonly used due to the high availability of terminal lines and the generality of the method.

6. Building Blocks of Personal Workstation Hardware

6.1 Physical Description

The personal workstation is normally constructed out of four items: a processing unit, a screen, a keyboard and a mouse. The processing unit stands on the floor but fits under a table. It will contain the central processing unit, the Winchester disk, the network connection and a variety of connectors for printers, terminal ports etc. This unit normally generates heat and incorporates a number of fans which make noise. These are often stored remotely from the user.

The screen is physically large, in order to support good windowing, with a very fine bit-map display. The windows are black and white and stand out from a grey, alternate black and white pixels, background. The keyboard incorporates a relatively large number of programmable function keys but otherwise is a normal layout. Beside the keyboard is a pointing device called a mouse used for graphical input or pointing.

6.2 The Seven M's

The performance of present personal workstations is judged by such things as processor power, memory size, screen size, disk size and network speed.

6.2.1 1 Mips.

The standard processor today is a Motorola 68010 or like a Motorola 68000 in performance. Personal workstations do exist with three times this performance. Future offerings include the Motorola 68020 with full 32 bit capabilities and the National Semiconductor 32032 which promises to be very fast.

6.2.2 1 Megabyte of memory

The memory is arranged to give about 1 Mbyte of real memory to the user. The resident part of the operating system will take up part of the real memory so it may be necessary to buy a 2 Mbyte machine in order to get such space for the user.

6.2.3 16 Megabytes of Virtual Memory

This is a function of the number of bits reserved by the microprocessor for addressing. New personal workstations exist with 256 Mbytes of virtual memory. It is expected that the limiting factor on the use of virtual memory will be the space allocated on the Winchester disk for page swapping.

6.2.4 Mega-pixel Screen

The bit mapped screens are physically larger than an A4 sheet of paper and have at least 1024x800 pixels. Each pixel is mapped onto memory taking part of the memory away from the user's program unless he directly wants to manipulate the screen. Fast hardware, called raster-op or bit-blt, is used to manipulate the screen.

6.2.5 10 Mbytes of Disk

This again is the amount of disk space allocated to the user. The system with its swap space can take up to 15 Mbytes of space making a 25 Mbyte disk necessary. Economically large disks are a better buy than smaller disks. This has led to the concept of a diskless workstation with a file server supplying disk space on the network. In this type of system multiple accesses to a single disk becomes a bottle-neck.

6.2.6 10 Megabit/second network.

The hardware for the 10 Mbit/second Ethernet is in place and most personal workstations work on networks of comparable speed. Token ring or slotted ring networks working in the 60 to 100 Mbit/second range are being produced and will be marketed in the near future.

6.2.7 Mouse

A mouse is a pointing device used for graphical input. The movements of a metal ball under the mouse can be sensed and the movements passed to the machine. A number of buttons on the mouse enable the pointing to be enhanced by clicking one of the buttons.

7. Building blocks of Personal Workstation Software

7.1 User Interface

The presence of fast hardware for handling the bit-mapped display and a graphical input device has made new user interfaces simple to construct. The main innovation is the presence of a display manager that directs output to windows and allows input via menu selection.

A window is a rectangular area of the screen reserved for input and output of a program. A program need not be restricted to one window, graphics programs often separate text and graphics into separate windows. In a multi-tasking system a number of programs may run concurrently each updating or accepting input from its own windows. This may be used for trivial things such as displaying the time or for complicated applications such as program debugging. In debugging the user would typically have a window open for the program, another for the debugger and at least one window viewing the original text of the program. Editing and recompilation can also take place in another window.

The menu system displays a menu of choices in a small window and the user may select options by pointing using a graphical input device such as a mouse. With a good menu system a user of a program need remember very little in order to run quite sophisticated operations. Much research is now being done into user interfaces using menus. The Macintosh represents a good implementation of a menu system where the names of the main menus are permanently on display across the top of the screen and the depth of menus is small. Menus that lead to other menus can lead a user through a labyrinth where he can easily become lost.

7.2 The Synthetic Camera

The ability to run many applications on one screen using good graphics has lead to new thinking in program design, especially, where graphics are directly concerned. The graphical representation of an object on the screen is a picture of some data in the computer that represents that object. This interpretation of the role of a program is to call the program a synthetic camera.

Traditionally, a graphics program would input data and by calls to a standard graphics package such as GKS [15] or CORE [16] represent that data graphically. Unfortunately while GKS and CORE can be used with bit-mapped displays they do not take advantage of their speed. Programs would therefore bypass these packages and directly manipulate the bit map themselves. Graphically the only common graphical representation of an object was the bit-map itself. Unfortunately a bit-map contains no information on the source of the bits. For example, is this a straight line, part of circle or even text? This makes scaling or manipulation of graphical objects impossible.

There has been a move therefore to concentrate on standard ways of representing data and then constructing programs as synthetic cameras to view this data. Object oriented programming is the name given to this type of programming. The objects or data representing these objects are stored in the system and are characterised by their long life. The objects are stored separately from the programs used to view this data. Finally, programs may use other programs to generate views of objects.

For example a text processing program may call upon one program to represent an object by a table, and a second program to represent the same object as a graph. Both the graph and the table are incorporated into a document. Changing the object will enable us to simply produce an updated document. In some systems this is done dynamically so that changes are immediately viewed on the screen within the document.

An example of such a system is 'Office by Example' [17] where data is represented in a standard way as tables in a relational database. The user has a small number of commands whereby he can select data from the database and display it in a variety of ways. A text processor and mail system is available to distribute the information.

Another example is Smalltalk [18] where objects are stored with the procedures and methods of manipulating them. A message to the object will perform one of the defined operations.

8. Recommendations on Personal Workstations in Europe

CERN has conducted some research into the personal workstations most suited to CERN's needs. We were able, along the way, to identify other areas where different personal workstations excel. For CERN the main requirement is for a powerful Fortran environment with a fast compiler, good symbolic debugger, fast editor for large files and a good performance analysis program. The table below shows a number of areas and one machine that excels in that area.

Table 1: Personal Workstations

Application Area	Machine
FORTRAN	APOLLO
CAD	APOLLO
UNIX	SUN
REAL-TIME	MASCOMP
ALGEBRA/LISP	SYMBOLICS 3600
CHEAP (BUT LIMITED)	MACINTOSH
IBM CULTURE	IBM-PC (and look-alikes)

Apollo has concentrated on CAD and has captured a large part of that market. Its machines are tailored for the large Fortran, graphical, programs typical of CAD. Their relatively wide range of personal workstations include integrated colour often required in CAD.

The SUN personal workstation runs the same version of the UNIX operating system as runs on the majority of VAX computers, a popular UNIX host machine. The communications between the systems are over Ethernet and use 'standard' TCP/IP protocols. Other personal workstations such as the Cadmus and the PERQ have implemented the Newcastle Connection.

Mascomp have modified their UNIX system for real-time work and their hardware to accept high data rates. Their version of UNIX quickly reacts to interrupts and tasks may be locked in memory. They also have a lot of experience in interfacing to equipment.

The Symbolics 3600 was developed at MIT to run the Algebra system MACSYMA which is a large program written in Lisp. The machine is adapted to the demands for high processing speed and large virtual memory required by Algebra systems.

The Macintosh is viewed by the personal workstation community as a big advance in personal computers. It is affordable by the individual and contains many of the essentials of a personal workstation. It is a rugged closed system. The closed nature of the system make it unsuitable for many applications.

Finally, the IBM-PC has few properties of a personal workstation. The open nature of the machine and the IBM culture that surrounds it has given it enormous popularity. The recent announcement of the IBM-PC AT introduces windowing in a general way showing that IBM is following the trends.

9. Recommendations on Buying a Personal Computer

Since most people will one day own a personal computer, the market place can only become more competitive at this low end of the range of single user systems. At this stage, the purchase of such a device can cost between \$150 and \$8000. A Macintosh would normally cost about \$2,500 and is made available to students at some universities for as little as \$1,100. While prices vary considerably it is necessary to judge the market place carefully before buying.

Some guidelines:

- a. Do not acquire a personal computer before you know what to use it for. This forms the basis of your judgement on the suitability of the personal computer.
- b. Look for clear definitions of what it can do in the areas that interest you. Good documentation based on clearly defined concepts is essential if you are to make effective user of the machine.

- c. Pay attention to the tools supplied for you to enter programs that you have either written yourself or copied from books or magazines. The speed at which you can do this will depend on the efficiency of the tools and the ease of program debugging.
- d. Avoid any system that is more complex than you can easily handle. At purchase time you should emphasize that you need a useful tool and not a fascinating toy.
- e. Be prepared for the fact that you will need educating in the use of personal computers. However intelligent the salesman says it is, it cannot replace your intelligence on its usage. Evaluate the educational material.
- f. Concentrate on the effectiveness of the personal computer's software and hardware and then take your decision.

10. Social Implications

There are three main areas of social importance. One is purely social and the other two are legal. Firstly, there is now a requirement for computer literacy which has created a new set of 'haves' and 'have-nots'. Secondly the copyright laws are proving as ineffective with software as they are with books, music and video. Finally, politicians must control databases.

10.1 'Have' and 'Have-nots'

The first clear distinction is that some people have personal computers and some do not. Further more, not everyone will be educated in their use. Computer networks give people access to a world of information which is denied others.

10.2 Copyright

The attempt to protect software pirates copying protected software has always failed since the pirates are as clever as the law abiding citizens. IBM software may be copied once and is 'impossible' to copy a second time, at least, using the IBM copy procedure. However a program is sold, legally, that overcomes this restriction. The sale is legal but the use is illegal. The author of this program has also protected his floppy against being copied by his own and by IBM's programs.

10.3 Databases

While politicians believe that little can be done about the first two problems they do believe that politicians must control databases.

In 1974 Sweden passed a law that all databases must be registered. France and Germany more recently applied similar laws. The law was conceived when the number of databases were small. The French body is reportedly overwhelmed with work and consequently can fulfil none of the useful tasks conceived for it. Sweden recently changed the law to exclude all databases that are for personal hobby use.

The U.S.A. government claims to control all financial databases. A small error in a financial database can cause personal problems. For example, a bank may refuse you a loan, a credit card company may refuse to give you a card, both, without the necessity to reveal the exact source of the information.

10.4 Conclusion

The industrial revolution enhanced our physical abilities. The computer revolution will enhance our mental capabilities. The English Luddites wrecked machines, cars kill, the third world is poor and the industrial nations rich. What will the computer revolution bring?

REFERENCES

- [1] Proceedings of the International Conference on Advanced Personal Computing Technology. CRAI, Italy. May 1984.
- [2] Corbato, F.J. and Vyssotsky, V.A. Introduction and Overview of the Multics System. AFIPS Conf. Proc., Fall Joint Computer Conf., vol. 27, pp 185-196, 1965.
- [3] Ada Reference Manual, U.S. Department of Defense, Jan 1983. ANSI/MIL-STD 1815A.
- [4] Wirth, Niklaus. Programming in Modula-2. Springer-Verlag. 1982.
- [5] C.P. Thacker, E.M. McCreight, B.W. Lampson, R.F. Sproull, D.R. Boggs. Alto: A personal computer. Xerox Palo Alto Research Center. Report CSL-79-11 (1979).

- [6] Levin, M., et al. The Lisp 1.5 Programmer's Manual, M.I.T., Cambridge, Mass. 1965.
- [7] MACSYMA Reference Manual. Mathlab Group, Laboratory for Computer Science, Mass. Inst. of Tech. 1977.
- [8] Wirth, Niklaus. The Personal Computer Lilith. ETH, Zurich.
- [9] B. Scheurer and N.Naffah. Local Networks and Individual Workstation in the Office of the Future and in the KAYAK project. French Swedish Symp. Data Proc. Telematics, Automation. Stockholm November 1980.
- [10] R. M. Metcalfe and D. R. Boggs, Ethernet: distributed packet switching for local computer networks, CACM (July 1976).
- [11] D.M. Ritchie and K. Thomson. The UNIX Time-Sharing System. Communications of the ACM, 17, No, 8 (July 1974), pp 365-375. The original UNIX paper.
- [12] Internet Protocol Transition Workbook. SRI International. March 1982.
- [13] Nelson, David L. and Leach, Paul J. The Evolution of the Apollo Domain. Hawaii International Conference on Systems Science, 1984.
- [14] D.R. Brownbridge, L.F. Marshall and B. Randell. The Newcastle Connection, or, UNIXes of the World Unite. Software Practice and Experience, Dec. 1982, pp 1147-1162.
- [15] Graphical Kernel System (GKS), Version 6.6. International Standards Organization. May 1981.
- [16] Status Report of the Graphics Standards Committee. Computer Graphics 13(3), August 1979.
- [17] Moshe M. Zloof. QBE/OBE: A Language for Office and Business Automation. IEEE Computer, May 1981.
- [18] Goldberg, Adele and David Robson. Smalltalk-80: The Language and its Implementation, Addison Wesley, 1983.

LIST OF PARTICIPANTS

LECTURERS

BLOBEL, V. Hamburg University, Fed. Rep. of Germany
BLOCH, T. CCVR, Ecole Polytechnique, Palaiseau, France
CANON, M.D. IBM Research Laboratory, San José, U.S.A.
CARRUBBA, F.P. Hewlett Packard Laboratories, Palo Alto, U.S.A.
CHURCHHOUSE, R.F. University College, Cardiff, U.K.
DELOBEL, C. Laboratoire IMAG, Grenoble, France
DOBINSON, R.W. University of Illinois at Champaign-Urbana, U.S.A.
and CERN, Geneva, Switzerland
HYAMS, B. CERN, Geneva, Switzerland
MIURA, K. Fujitsu Ltd., Kawasaki, Japan
RANDELL, B. The University of Newcastle, U.K.
ROSNER, R.A. London University Computer Centre, U.K.
RUBBIA, C. CERN, Geneva, Switzerland
SALICIO, J. DESY, Hamburg, Fed. Rep. of Germany
SANTIAGO, S. CERN, Geneva, Switzerland
VERDEJO, M.F. Facultad de Informatica, Donostia, Spain
VERGES, M. Universidad Politecnica de Barcelona, Spain
VON RÜDEN, W. CERN, Geneva, Switzerland
WIEGANDT, D. CERN, Geneva, Switzerland
WILLERS, I. CERN, Geneva, Switzerland

* * *

STUDENTS

ALVAREZ, P.A. Universitat Autonoma de Barcelona, Spain

BAILLIE, C. Edinburgh University, U.K.

BALAND, J.F. Mons University, Belgium

BATLLE, J. Universitat Politecnica de Catalunya, Girona, Spain

BEHRENS, M. DESY, Hamburg, Fed. Rep. of Germany

BERNARDI, E. Bonn University, Fed. Rep. of Germany

BESSET, D. Princeton University and SLAC, Stanford, U.S.A.

BRUN, R. CERN, Geneva, Switzerland

BURKIMSHER, P. CERN, Geneva, Switzerland

CALLIGARO, T. CRN, Strasbourg, France

CALVINO TAVARES, F. Universitat Autonoma de Barcelona, Spain

CAMPBELL-BURNS, P. University College, London, U.K.

CASANOVAS, J. Facultat d'Informatica, Barcelona, Spain

CASTANO, B. JEN, Madrid, Spain

CASULLERAS AMBROS, J. Facultat de Ciencies, Bellaterra, Spain

CHEVALLIER, J.-M. Ecole Polytechnique, Palaiseau, France

CHRISTIANSEN, H.P. CERN, Geneva, Switzerland

COLINO, N. JEN, Madrid, Spain

CRESPO, J.M. Universitat Autonoma de Barcelona, Spain

DAVERVELDT, P.-H. Instituut-Lorentz voor Theoretische Natuurkunde,
Leiden, Netherlands

DE BIE, J.E.P. NIKHEF, Amsterdam, Netherlands

DIEZ, F. JEN, Madrid, Spain

DORENBOSCH, J. NIKHEF-H, Amsterdam, Netherlands

DÜ, S. LAL, Orsay, France

EDMENDS, J. University College, London, U.K.

ENDERBY, M. Daresbury Laboratory, Warrington, U.K.

FARILLA, A. Bari University, Italy

FLYNN, P. Rutherford Appleton Laboratory, Chilton, U.K.

GAISFORD, P. SIN, Villigen, Switzerland

GARRIDO BELTRAN, L. Universitat Autonoma de Barcelona, Spain

GATO RIVERA, B. Instituto de Estructura de la Materia, Madrid, Spain

GHINET, F. CERN, Geneva, Switzerland

GONZALEZ ROMERO, E. JEN, Madrid, Spain

GUSTAFSSON, L. CERN, Geneva, Switzerland

HAUSAMMANN, R. Geneva University and CERN, Geneva, Switzerland

HERR, W. Heidelberg University, Fed. Rep. of Germany

HORISBERGER, R. CERN, Geneva, Switzerland

JEFFERY, D. Bristol University, U.K.

KOSTARAKIS, P. CERN, Geneva, Switzerland

LEVY, T. London University Computer Centre, U.K.

LINDERS, J. CERN, Geneva, Switzerland

LLABERIA GRINO, J.M. Universitat Politecnica de Barcelona, Spain

LLAMOSI, A. Facultat d'Informatica de Barcelona, Spain

LOPE ANTON, P. JEN, Madrid, Spain

MARECA, M.P. Instituto de Estructura de la Materia, Madrid, Spain

MARIN, J.C. CERN, Geneva, Switzerland

MARINIER, J.-P. CEN Saclay, Gif-sur-Yvette, France

MARROCCHESI, P. INFN, Pisa, Italy

MARTINEZ LASO, L. JEN, Madrid, Spain

MARTINEZ RODRIGUEZ, M. Universitat Autonoma de Barcelona, Spain

MATO VILA, P. Universitat Autonoma de Barcelona, Spain

MATTHIESSEN, U. Dortmund University, Fed. Rep. of Germany

MCCLATCHY, S. CERN, Geneva, Switzerland

MCCLATCHY, R. Sheffield University, U.K.

MCKEEMAN, A. CERN, Geneva, Switzerland

MOLLAR, M.D. Facultad de Fisica, Valencia, Spain

MOLLINEDO DE LAS HERAS, A. JEN, Madrid, Spain

MORALES, J. Zaragoza University, Spain

NEYROUD, N. LAPP, Annecy, France

NORTH, N. National Physical Laboratory, Teddington, U.K.

NYGARD, M. CERN, Geneva, Switzerland

OEHLSCHLÄGER, J. Kernforschungszentrum Karlsruhe, Fed. Rep. of Germany

OPPENHEIM, R. Yale University, New Haven, U.S.A.

OSEN, K. CERN, Geneva, Switzerland

PASINI, P. INFN, Bologna, Italy

PRAT-MARTINEZ, J. Laboratoire de Physique Corpusculaire de
Clermont-Ferrand, Aubière, France

RANITZSCH, K.H. Kernforschungszentrum Karlsruhe, Fed. Rep. of Germany

RAUCH, W. Lawrence Berkeley Laboratory, U.S.A.

REINICKE, P. Bonn University, Fed. Rep. of Germany

RODRIGO, T. JEN, Madrid, Spain

ROTSCHEIDT, H. Bonn University, Fed. Rep. of Germany

ROUSSILLE, M. Mons University, Belgium

RUETSCHI, A. Institute of Medium Energy Physics, Villigen, Switzerland

SAINZ, M. Escuela Universitaria Politecnica, Girona, Spain

SALDANA, F. CERN, Geneva, Switzerland

SCHWABL, W. Technical University, Vienna, Austria

STANESCU, C. INFN, Rome, Italy

TOPPANO, E. Udine University, Italy

VANDE VYVRE, P. CERN, Geneva, Switzerland

VERKERK, P. Université Pierre et Marie Curie, Paris, France

VIDAL, M. Max-Planck-Institute, Munich, Fed. Rep. of Germany

VILLAR, J.A. Zaragoza University, Spain

WENDT, K. CERN, Geneva, Switzerland

WEYMANN, M. Albert-Ludwigs-University, Freiburg, Fed. Rep. of Germany

WU, W.-M. Institute of High Energy Physics, Beijing, The Peoples
Republic of China

ZELUDZIEWICZ, T. Institute of Nuclear Physics, Cracow, Poland