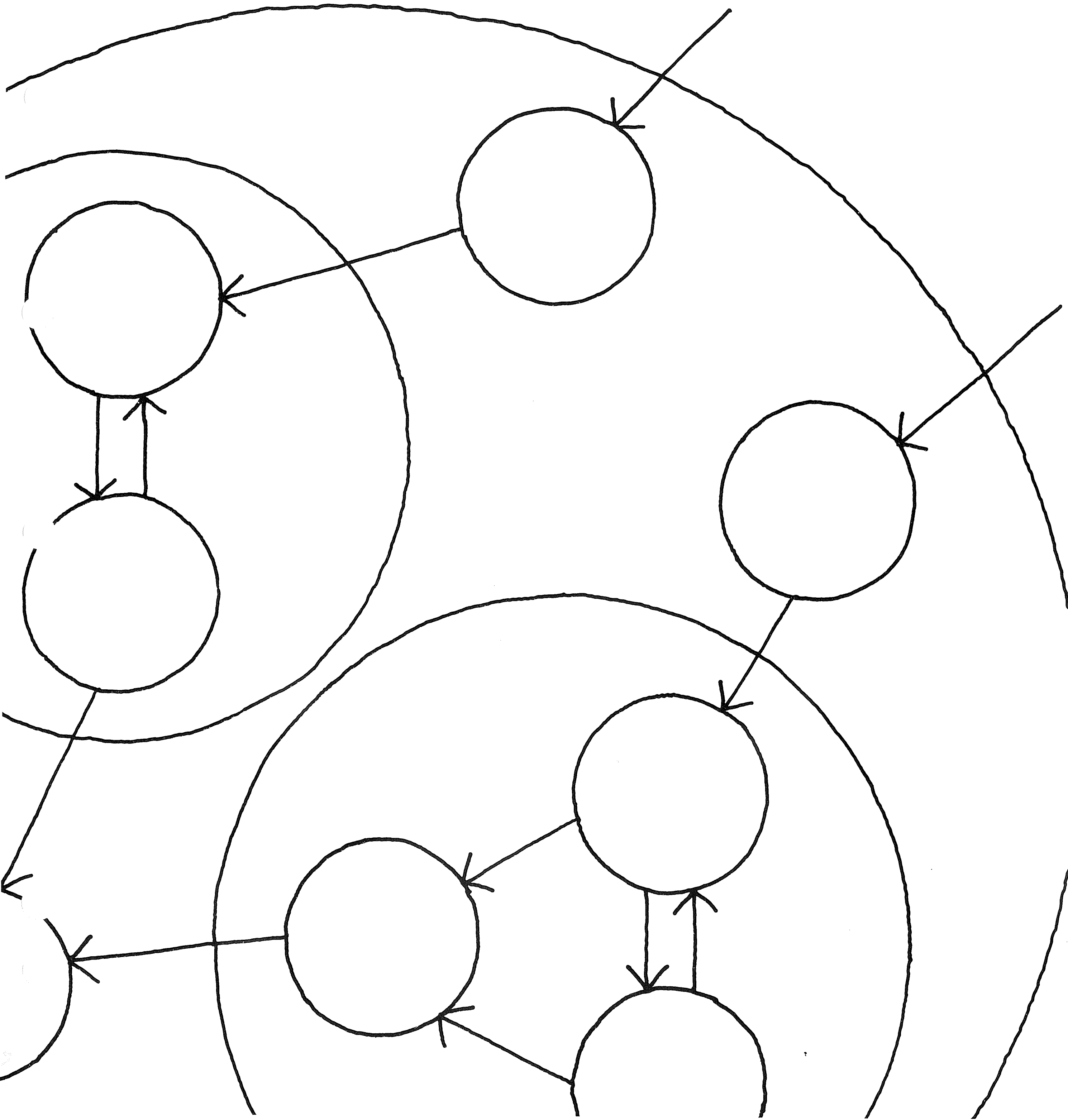
Language Overview

occam®



Entia non sunt multiplicanda praeter necessitatem
William of Occam

Occam's Razor – that entities are not to be multiplied beyond necessity – has been the guiding principle in designing this new language. William of Occam was a fourteenth century Oxford philosopher, whose teaching was condemned by the Pope. He is now recognised as having anticipated a basic principle of modern scientific method.

The choice of what is to be omitted from a new language is in practice much more critical than the choice of what is to be added
Niklaus Wirth

Niklaus Wirth is renowned as the designer of Pascal. Together with Dijkstra and Hoare, he has been influential in establishing the principles of good programming practice. Ada, the most ambitious language development ever attempted, is largely based on Pascal. Wirth's quote is from the original 'Green' submission to DoD for the Ada contest. Green won, but Wirth's comment was omitted from the final version.

Programmers are always surrounded by complexity; we cannot avoid it. If our basic tool, the language in which we design and code our programs, is also complicated, the language becomes part of the problem rather than part of its solution
C.A.R.Hoare

Professor Hoare, Director of the Programming Research Group at Oxford University, is well known for his concern over the unnecessary elaboration of languages. He fears that systems programmed in complex languages may pose dangers in the real world. He received the Turing Award – the ACM's highest honor for technical contribution to the computing community – for his 'fundamental contributions to the definition and design of programming languages' with work 'characterised by an unusual combination of insight, originality, elegance and impact'.

Tony Hoare has been closely involved in the design of occam, the new language developed at INMOS by David May to provide a better tool for programming microprocessors and future systems. The precursors of occam are well structured languages like Pascal and C and experimental languages like Hoare's CSP – which established the communication primitives subsequently elaborated in Ada.

Sequential systems will not be adequate for the future. There are an additional four orders of magnitude in computational capability available through concurrent systems
Carver Mead

Carver Mead is the foremost advocate of structured VLSI design. He holds the chair at Caltech endowed by Gordon Moore, Chairman of Intel.

Concurrency is clearly the key to higher performance systems. Occam is designed to unlock the potential of VLSI by providing the concepts for describing and programming systems containing many interconnected processing elements – the fifth generation systems of the future.

Occam is the new programming language.

Occam is based on the concepts of concurrency and communication. These concepts enable today's applications of microprocessors and computers to be implemented more effectively. They are essential to tomorrow's systems built from multiple interconnected transputers.

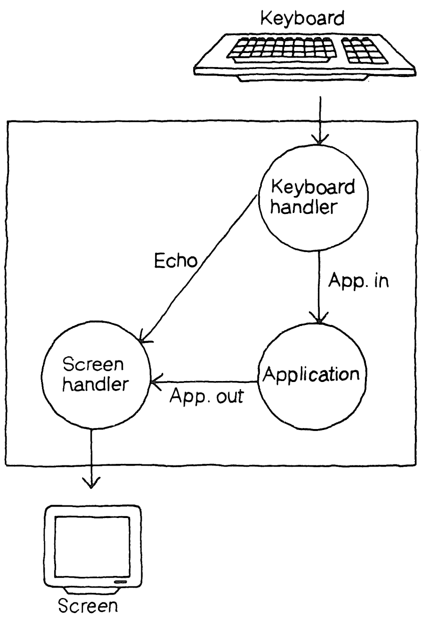
Occam is designed for programmers and engineers. The language is oriented towards interactive use. It enables complex systems to be described in a concise and readable form. As a result, programmer productivity is enhanced.

Occam has a formal basis and uses the minimum of concepts. It is easy to understand and efficiently implemented on a wide range of microprocessors and computers.

The use of occam is illustrated by the design of a simple interactive system, concentrating on the design of a screen handler. The top level description is that of a system reflecting the characters typed on a keyboard, and displaying the results of an application process. It also includes a timeout which prompts for input.

The system is implemented by three distinct tasks, one to accept characters from the keyboard, one performing the application, and one to merge the results and the reflected characters and output them to the screen.

These tasks can be represented as a network.



In occam, each task is represented by a process, and each connection by a channel. The processes communicate by sending messages via the channels. A process can be constructed from smaller processes. Indeed this collection of processes is itself a process in occam, and the system as described could be part of some larger system.

This network is represented by defining the channels and processes. **CHAN** introduces the channels through which the processes communicate, and the **PAR** construct causes the various processes to operate concurrently. Each process is independent and so the three processes can execute in parallel:

```
CHAN Echo, App.in, App.out:
PAR
  ... keyboard handler
  ... application
  ... screen handler
```

If this were a large system, then each component process could be given to an individual programmer or team to design.

The screen handler process may do one of three things. Firstly, it may receive a character from the keyboard handler, which it reflects to the screen:

```
Echo ? ch
  Screen ! ch
```

Secondly, the screen handler process may receive a character from the application. The **IF** construct is used to test the character to see if the whole process should be terminated:

```
App.out ? ch
  IF
    ch = terminating.character
      running := FALSE
    ch <> terminating.character
      Screen ! ch
```

Finally, at five second intervals, the screen handler process outputs a prompt:

```
TIME ? AFTER alarm.time
  Screen ! '>'
```

These individual program sections are combined into the complete screen handler process by declaring the local variables, by using **WHILE** and **ALT** to enable the process to perform whichever alternative is required, by using a timer input to reset the alarm clock immediately before waiting for a message from one of the two channels, and by using **SEQ** to control the initialisation of the variables:

```
VAR ch, running, alarm.time :
SEQ
  running := TRUE
  WHILE running
    SEQ
      TIME ? alarm.time
      alarm.time := alarm.time + five.secs
      ALT
        Echo ? ch
          Screen ! ch
        App.out ? ch
          IF
            ch = terminating.character
              running := FALSE
            ch <> terminating.character
              Screen ! ch
        TIME ? AFTER alarm.time
          Screen ! '>'
```

Entia

Model	Programs are expressed in terms of concurrent processes, which communicate using channels. An obvious implementation of an occam program is a network of microcomputers, each executing one of the concurrent processes. However, the same occam program can also be execured by a single computer sharing its time between the concurrent processes.
Values	Values correspond to numbers, characters, truth values and bit patterns. Arrays are provided. There is a wide range of logical and arithmetic operators for use in expressions.
Structure	Programs are contructed from a small number of primitive processes: assignent, input and output. Processes are combined using the constructors sequence, parallel, conditional and alternative, and may be replicated in arrays.
Assignment	An assignment may be used to set the value of a variable to the value of an expressions.
Communication	A channel provides communications between two concurrent processes. The communication is synchronised, and takes place only when both the input and the output process are ready; the values being copied from the output process to the input process.
Time	Execution of a process may be related to the passage of time. A timer input may be used to delay execution until a specified time is reached.
Sequence	The component processes are executed one after the other. A sequence construct terminates after the last of its components has terminated.
Parallel	The component processes are executed concurrently. Each component process operates on its own variables, communicating with other concurrent processes using channels. A parallel construct terminates only after all of its components have terminated.
Conditional	The component processes are tested in sequence. If one is ready, it is executed. At most one of the component processes is executed.
Alternative	An alternative waits for input from one of a number of channels, and then executes a corresponding component process.
Repetition	A while construct causes its component process to be executed repeatedly until the result of evaluating a condition is false.
Procedure	Procedures may be defined, and may take channel parameters, allowing a form of process abstraction. A completely self–contained form of abstraction is provided. This is an independent unit of compilation, and may be transmitted around a system, or loaded when a system is initialised.
Syntax	Each primitive process and each constructor is represented by a single line of program. The component processes combined by a constructor follow it on successive lines. This makes interactive editors and compilers simple and efficient.

Semantics

The design of occam is based on a formal model which facilitates reasoning about the properties of the language constructs, and the behaviour of specific programs. Each process can be described by an assertion in the predicate calculus, and the composition of processes into networks can be described by the logical conjunction of the assertions describng each process.

Use of Occam

Since its introduction in 1982, occam has been widely used for many purposes, including :

Programming concurrent systems containing hundreds of processors;
Systems programming, operating systems and compilers, eg the occam programming system;
Real time industrial control systems;
Signal processing and image processing;
Teaching the principles of concurrent and real time programming;
Hardware description, eg as an HDL for the transputer;
Simulation, eg electronic systems, manufacturing systems.

Occam has been successfully used in the construction of large systems by teams of programmers. This has exploited the capability in occam for an interconnected set of processes to be regarded from the outside as a single process. At any level of detail, each individual designer is only concerned with a small and manageable set of processes.

Products

The occam language is supported by a wide range of products from INMOS. These include introductory products for language evaluation, integrated support environments, and portable versions of the compiler.

Publications

Occam Programming Manual *(English)*	Prentice Hall International ISBN 0–13–629296–8
Occam Programming Manual *(Japanese)*	Keigaku ISBN 4–7665–0133–0

Occam user group

The occam user group is an informal organisation run by its own members. Its main aim is to act as a forum for the interchange of information among members and as a channel for communication with INMOS. It organises meetings twice yearly, and issues a newsletter, also twice yearly. Membership is free upon submission of an enrolment form, available from Software Support at INMOS.

inmos®

INMOS Limited
Whitefriars
Lewins Mead
Bristol BS1 2NP
England
Telephone (0272) 290861
Telex 444723

INMOS GmbH
Danziger Strasse 2
8057 Eching
Munich
West Germany
Telephone (089) 319 10 28
Telex 522645

INMOS Corporation
PO Box 16000
Colorado Springs
CO 80935
USA
Telephone (303) 630 4000
TWX 910 920 4904

INMOS SARL
Immeuble Monaco
7 rue Le Corbusier
SILIC 219
94518 Rungis Cedex
France
Telephone (1) 687 22 01
Telex 201222

September 1985
72-OCC-027-000