My DCS file

SCIENCE & ENGINEERING RESEARCH COUNCIL

RUTHERFORD APPLETON LABORATORY
COMPUTING DIVISION

D I S T R I B U T E D   C O M P U T I N G   N O T E   5 7 2

VISITS

issued by
Dr D A Duce

Notes on a visit to Dr J Darlington
Imperial College 5 February 1982.

15 February 1982

DISTRIBUTION:	R W Witty
	D A Duce
	Miss G P Jones
	F R A Hopgood
	J Monniot
PRESENT:	Dr I Watson, Manchester
	Dr D A Duce
	Dr J Darlington, Imperial College
	M Cripps, Imperial College
	V Wu. Imperial College
	I Moor, Imperial College
	M Reeve, Imperial College

1. INTRODUCTION

The visit formed part of a Panel visit to Dr Darlington to discuss a proposal from Dr Darlington to construct a general purpose application language machine and applicative programming environment. The formal Panel visit is to take place on 8 February, but Dr Watson being unable to attend on that day kindly agreed to visit Dr Darlington beforehand and to relay comments to the Panel through the Academic Coordinator.

The Visiting Panel will formally report through the Chairman of the DCS Panel to the CCSC Meeting 16/17 February. Professor Needham was invited to be a member of the Panel but had prior commitments on all possible dates for the meeting. He was however able to discuss the application with Dr Darlington on 3 February.

The application was first considered by the DCS Panel in October 1981. There was no doubt as to the scientific excellence of the proposed project but Panel Members were concerned about the ability of the group to undertake what looked to be a major constructional project. The Visiting Panel were specially charged to investigate this aspect.

2. BACKGROUND

Dr Darlington gave an overview presentation of the project.

Five people are directly associated with the project:

John Darlington	
Martin Cripps,	Senior Lecturer and Director of Wolfson Microprocessor Research Unit
Victor Wu,	RA
Ian Moor,	Teaching Assistant
Mike Reeve,	Final Year PhD Student

Curricula vitae were provided. The project crucially depends on the excellence of the people engaged in the work. Mike Reeve (one of best Firsts for a long time) and Victor Wu both have hardware experience, Mike worked with IBM during his vacations on various hardware projects and received an IBM Special Contribution Award.

A number of people also contribute to the project:

Tony Field,	PhD Student (electronics background) Networks, Load Sharing.
John Goodchild,	PhD Student (first year) Expert Systems for Program Development
Sue Eisenbach,	Microprocessor Support Unit Westfield College,
Roger Bailey,	Lecturer

In addition the whole of the Logic Group (Bob Kowalski, Keith Clark, Krysia Broda and others) cooperate with Dr Darlington's project on a broad front. The Logic Group regard cooperation as essential and fruitful.

The present grant application is seen as the development of an integrated applicative programming environment and ultimately as a solution to the 'software crisis'. Applicative languages are particularly pertinent to this problem as they are side effect free, have clean mathematical properties, allows software to be developed by transformational techniques (making proof much simpler) and admit the possibility of parallel evaluation.

The work of the group is based principally on the language HOPE which is a higher order recursion equation based, strongly typed, functional language. The precursor of HOPE was NPL developed at Edinburgh by Rod Burstall and John Darlington, which grew out of work on program transformation.

As an example of HOPE, consider the factorial function. This may be written in HOPE as:

```
                         fact (o) <=1
                         fact (n+1) <= n+1  fact n
```

New date structures can be defined by constructors, eg list:

```
                         append (nil,y) <=y
                         append (a::x,y) <=a::append (x,y)
```

(::is an infix operator for cons)

As an example of higher order functions consider:

```
                         map (nil,f) <=nil
                         map (a::x,f) <= f(a)::map(x,f)
```

map applies f to every element of a list. The first equation specifies the action for a null list, the second for a list of the form a consed onto x. A case of map is in the function double, which doubles every element of a list:

```
                         double (l) <= map (l, lanbda x => 2.x)
```

The language supports strong polymorphic typing as developed by Milner, for example:

```
                         data tree (alpha) = niltree
                                                  ++ constree (alpha,
                                                         tree (alpha)
                                          tree (alpha))
```

where alpha is a type variable

HOPE is claimed to be a usable functional language, for example it includes:

```
                         modules
                         local variables
                         infix, distfix operators
                         lazy     evaluation     (allowing     infinite
datastructures to be
                                          manipulated)
```

(A distfix operator is a general name with holes for operando, eg if-then-else). Tools and anenvironment to support HOPE are building up. A compiler for HOPE, written entirely in HOPE has been written which generates Alice Compiler Target Language (CTL). Alice is the name given to the applicative language machine.

The construction of the compiler was very illuminating as a software engineering task. It was easy to construct and get right first time. Higher functions were absolutely essential for this task, as were types. The compiler essentially works by definin a HOPE object program as a datatype of HOPE and then defining functions which walk over this structure. It is a highly parallel multiphase compiler. The code size is roughly 3,500 lines of HOPE. A paper summarising this work has been accepted for the Boston Compiler Conference; copies were provided for the Panel.

The compiler has been implemented on the Edinburgh DEC10. The code files produced can be transformed over SERCnet and Metronet to the IBM machine at Imperial College where they can be run on an Alice simulator (see later).

The compiler also illustrates another benefit of the applicative language approach in that by changing cons to lazy cons, a good improvement in space utilisation was achieved without affecting the correctness of the compiler. Work is in hand to bootstrap the HOPE compiler to a Pascal compiler; currently HOPE runs over POP2.

V Wu has written a Pascal HOPE interpreter. This is a straightforward HOPE source interpreter which borrows ideas from Alice. The system has been used in teaching applicative languages and has stood up extremely well. The interpreter supports higher order functions and lambda expressions; type checking is currently being added.

Sue Eisenbach at Westfield College is developing a microprocessor based HOPE system. This compiles to an abstract 3 address machine and runs on an 8-bit microprocessor. This is not seen as a mainstream activity.

Collaboration with the Logic Group is very fruitful. Currently there exist designs for parallel implementations on Alice of:

(i) the relational language of Clark and Gregory hopes to use applicative anaolgue of Dijkstra6s "don't care" non determinism (once made a choice cannot backtrack over it)

(ii) full PROLOG with and/or parallelism (Broda)

In addition the SERC funded project of Clark and Gregory hopes to use both the Manchester Dataflow Machine and the Alice Simulator.

Program transformation work is also in hand. The nutaon if program transformation lies at the heart of the systematic development if correct efficient software and forms an essential part of the environment:

(i) write clear initial version of program ignoring questions of efficiency;

(ii) systematically transform to a more efficient version using transformations guaranteed to preserve correctness.

The basis of this work is the unfold/fold operators developed by Burstall and Darlington. Application of the operators is governed by six rules, guaranteed to pressure (patial) correctness.

This has lead to the development of the Metalanguage Transformation System by Darlington, Moor and Wu, cf Milner's LCF project. The user can explain to the system how he wants the transformations to be carried out. This is done using a metalanguage to describe transformations in a structured way in terms of unfold and fold. This system is novel in that HOPE is used as its own Metalanguage, rather than using a separate Metalanguage as in LCF.

Work is now commencing on higher level tactics (in the LCF terminology) which are built out of the first level, and involve limited search to achieve their goals and use Hope modules for security. They are guaranteed never to be wrong. Examples of use would include the removal of recursion, change of data type, merging loops etc.

This work was written up as an invited paper for the Amsterdam Conference on Algorithms, October 1981.

Dr Darlington summarised the group's plans as:

(i) Complete Hope compiler;
    type checking
    evaluation strategies (stated separately from text and giving control over use of resources and way to go about computation)
    modules

(ii) develop PROLOG Compiler

(iii) develop applicative programming tools
    (a good student is developing A HOPE structure editor)

(iv) develop transformation system, particularly higher level tactics and application to significant programs

(v) build Alice

3. Alice

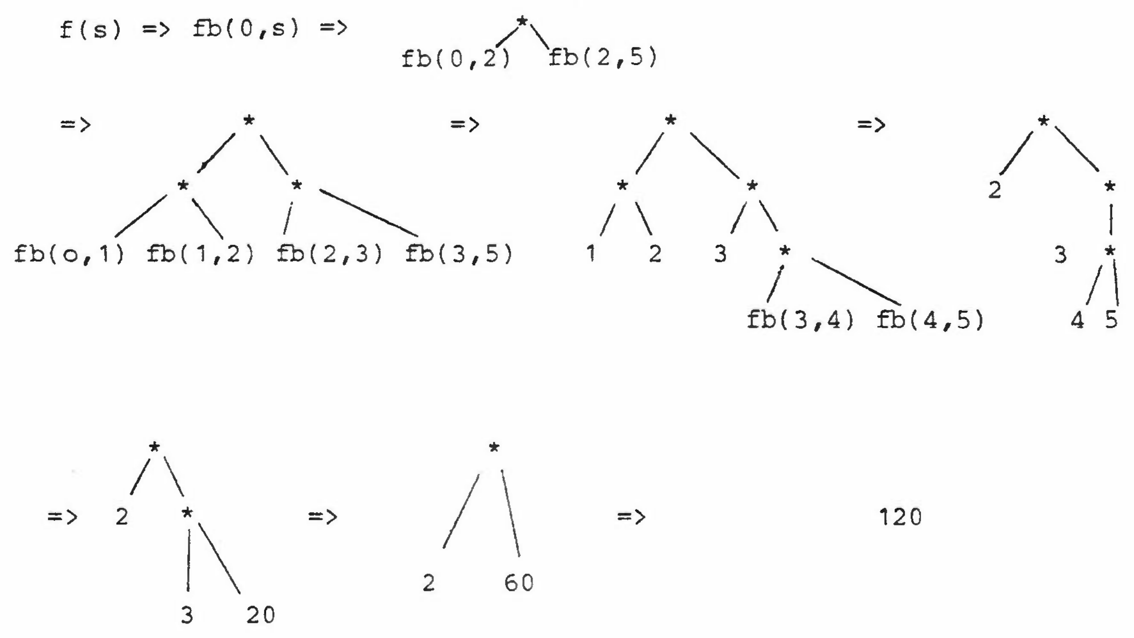
Mike Reeve; then gave a detailed presentation of Alice.

The motivation behing Alice is that applicative languages are a good thing but traditional implementations are inefficient. The approach taken is Alice is based on graph reduction.

Consider the binary algorithm:

```
f (n) <= fb(o,n)
fb(i,i(<=i
fb(i,i+l
fb(i,j)<fb(i,mid)*fb(mid,j) where mid = integer divide (i+j,2)
```

The evaluation of f(5) can be viewed as a series of reductions applied to the expression represented as a graph:



This can be modelled using packets, stemming from the Manchester dataflow archeticture. A packet has the format:

ID	FUNC	ARG LIST	STATUS	SIGNAL LIST	REF COUNT

ID is a unique name for the packet, FUNC is the function name (*,f,fb etc) and ARGLIST the list of arguments. The other fields are used by the evaluation mechanism.

The above computation could thus be represented as:

```
i      f       x
    integer
    literal
       ⇓
i      fb     [o]   [5]          where [n] is the identifer of a packet
                                 containing the integer literal n
       ⇓
i      *       j     k       -A

j      fb     [0]   [2]  which can run in parallel
k      fb     [3]   [5]
       ⇓
      etc
```

At point A the computation cannot proceed until the computation ofpackets of j and k is complete. Rather than have the processor enter a busy wait state, a processor picking up packet i will note the interest in packets j and k and will place the identifier of i in the signal fields of packets j and k, and i is marked 'asleep'. When j and k have been rewritten, i is awoken.

The evaluation scheme presented is eager evaluation. There is a user definable tag in the packet allowing eager of lazy mode evaluation which permits infinite data structures to be supported. Consider:

```
cond (P,Q,R)
```

'cond' is conditional evaluation of Q or R as P is true or false. Eager evaluation in this situation is not always safe (Q or R may not terminate). Lazy evaluation is always safe, but inefficient when eager evaluation. The user may write:

```
cond (P, "suspended" Q, "suspended" R)
cond (TRUE,Q,R) <= "activate" Q
cond (FALSE,Q,R) <= "activate" R
```

Note that user in this context may be the system (compiler say).

Natively input/output could be defined as:

```
printlist (x::L) <= print (x):: printlist (L)
```

The problem with this definition is that one cannot guarantee execution in the right order. The correct definition which preserves sequencing is:

```
printlist (x::L) <= print (x,X)::X

where X="suspended" printlist (L)
print (x,X) <= send to vdu, "activate"x
```

Packets can be treated as variables, with assignment at the machine level which provides for logical variables, in place updating (an optimisation) and implementation of imperative languages (sequencing allows full von Newman style).

First order functions are supported directly, higher orderly on the fly function definition. Turner's SASL can be supported and in fact Turner's Combinator machine has been implemented in Alice CTL (about two and a half pages). David Turner is hoping to move his SASL system into Alice.

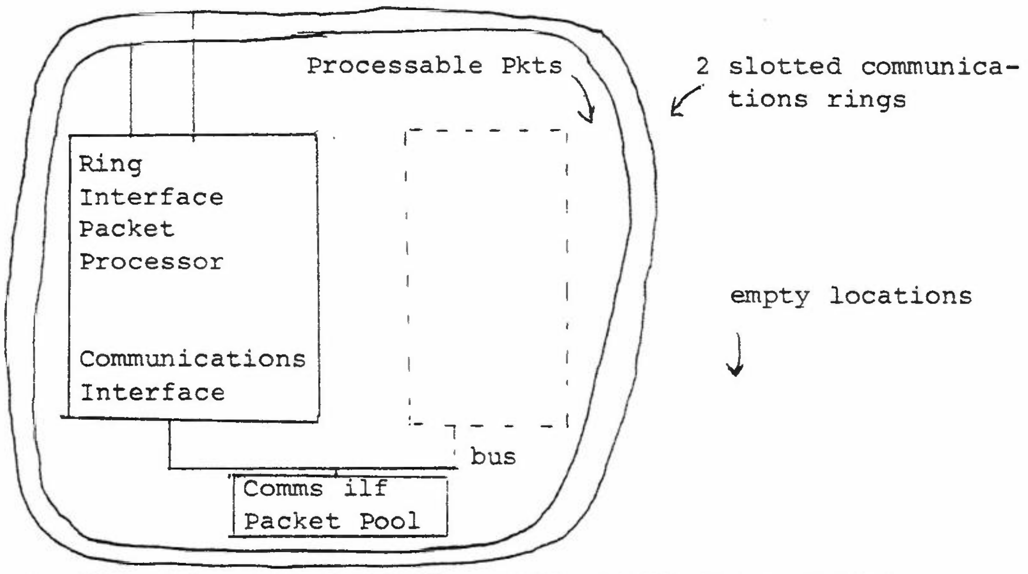
Languages with "don't care" non-determinism can be supported. A variety of arbitration methods are available for deciding which rewrite rule applies. Alice is really a production system in the classical AI sense. Logic languages can be supported via connection graphs or direct manipulation of and/or trees.

3.1 Abstract Architecture

The abstract architecture of the Alice machine is a set of computing agents accruing a common pool of packets. If all the required arguments of a packet are available, the packets corresponding to the right hand side of the rewrite rules will be generated and put into the pool. Garbage collection is done by reference count.

The computing agents are packet rewrite engines (look up function definitions in table and produce appropriate packets) and are fairly straightforward. It is the packet pool that is difficult.

A more concrete architecture is the following:



This is the desk top machine. The outer ring contains unused indentifiers, the inner ring identifiers of rewriteable functions. When an agent is free it picks an identifier from the inner ring. When an agent wants to produce a new packet it takes an identifier from the outer ring.

A performance estimate may be given:

Assume agent rate is 128 sec per rewrite
packet pool rate 1 sec per transaction

Approximately 6 packet pool accesses taken per rewrite (from simulations)

$\frac{128}{6 \times 1}$ 20 agents before real bus contention problems arise

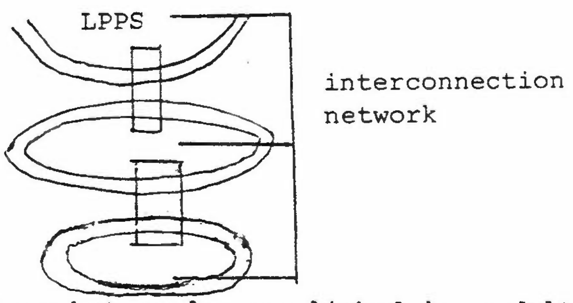
Effective system rate

$\frac{128}{20}$ sec per rewrite macro 6 sec

170K rewrites per second

There is a scale up factor of 15-25 to the equivalent von Newman rate, so the approximate equivalent is two and a half Mips. This is about 2 orders of magnitude better than HOPE on the DEC10.

Bigger machines can be built by interconnecting desk top machines:



Local packet pools are linked by a delta network to provide a global address space across all machines. A delta network performs best with a large number of nodes and random accesses (better than localising references).

A performance estimate for a machine with 4K ports (roughly 8 agents per port) is about 100M reductions per second.

3.2 CTL

CTL is an intermediate code for compilers and a high level assembler. It forms an implementation independent stable base for the software projects.

The execution cycle of Alice is:

1. extract processable packet
2. decide whether it is rewriteable if not leave signal requests in argument packets
3. determine rewrite rules to be applied
4. increment appropriate reference counts
5. generate packets representing RHS
6. deposit RHS packets in pool
7. decrement appropriate reference counts
8. go to 1

Consider an example:

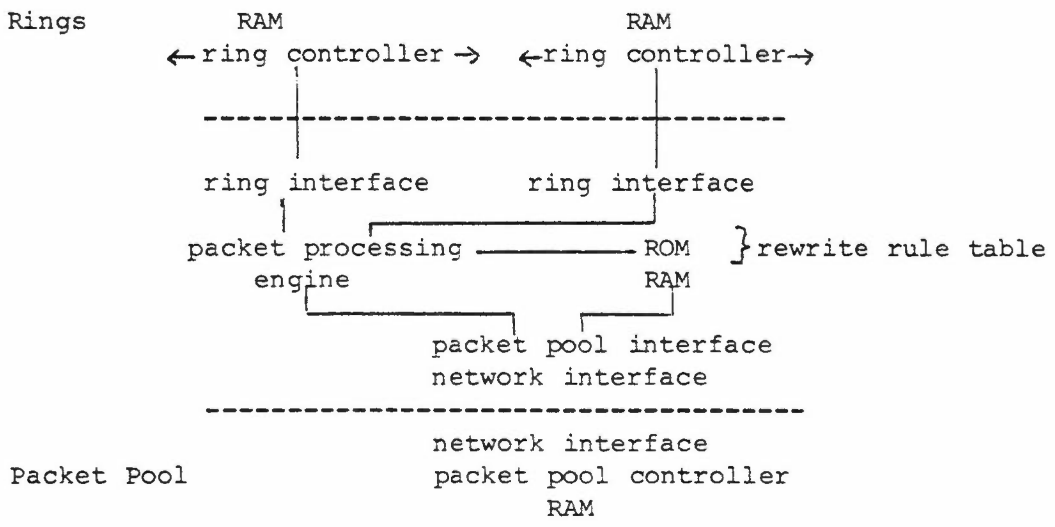
```
data list (alpha) = = nil
                      + + cons (alpha, list (alpha))

dec  append: list (alpha) x list (alpha)  list (alpha)
---  append  (x,nil)
             <=x
---  append  (x, cons (a,l))
             <= cons (a, append (l,x))
```

This generates the CTL code:

```
     Constructor nil, cons
     rewriteable append
     rules for append
          requires arg (2)
          rule arg (2). function = nil
          rhs
          @ synonym (arg(1))
          negative-deltas arg (2):-1
     rule arg (2). functions=cons
          rhs
          1 append (arg 1 arg(2) arg(2))
          @ cons (arg(2) arg(1) &1)
          negative-deltas arg(2):-1
     and rules
```

The 'rules' clauses behave as guarded commands. Any number of 'requires' clauses are permitted which may be disjoint.

3.3 Concrete Architecture



The slotted communications rings are implemented as stacks. When an agents local stack is exhausted or overflows, it will propagate its request to its neighbours stacks. Optionally intelligence in ring controller can spread load in idle time.

The architecture is modular and suited to IC implementation. It is feasible to implement functional units in LSI with smooth transition to VLSI. The design can be made homogeneous at the sub-component level and Mike Reeve has discovered a way to build the machine from a single type of chip, called a Champion (cf Baron's transputers).

A Champion consists of:

- finite state machine controller
- ACU
- Scratch pad RAM
- 2 x 16 bit bi-directional ports (vertical links - 2 control per port)
- 2 x 8 bit bi-directional port (horizontal links - 4 control lines per port) also use to do bus arbitration on vertical links)

This can be packaged in a 64 pin package:

i/o ports	60 pins
interrupt	1
clock	1
power	2
	64

The finite state controller has to be custom designed for each unit but the other components are standard across all units in the machine and also should be readily available from a component library.

Mike estimates that refinement of the design of the single chip building block will take about 9 months. Writing of the microcode can take place in parallel with this. It is anticipated that the LSI implementation of the chip will be complete 2 years after the start of the grant, the first trial chip should be ready after 18 months. Then construction and commissioning of the desk top Alice can commence.

Mr Portman asked how much memory was allocated to the ring RAM6s.

Mike Reeve replied that the prototype will have a packet pool of 64-128k packet and the ring RAM6s will be 2-4K. The packet pool controller will be able to perform high level access to the pool, for example get the second argument field of a given packet or is a given packet a constructor function. The packet pool interface also has intelligence and can do some caching. A packet may safely be cached if the reference count is one as then no other agent can be using it.

Mr Portman asked how many Champions are needed to build the machine. There are 8 chips per agent and 2 for the packet pool: thus about 200 working chips are necessary.

3.4 Current State

The Compiler Target Language, the central theme of the project is stable. A HOPE to Alice CTL compiler and Alice CTL to machine code translator have been implemented. A simulator for Alice at the functional unit level is operational (written in Pascal). A register transfer level simulator (written in PATH Pascal) was written but has been made redundant by Mike Reeves' realisation that the machine could be constructed from one type of superchip rather than the 6 different types originally envisaged.

Work in progress includes:

```
     extensions to simulator
                   caching
                   large scale machine
     VLSI implementation of desk-top machine
     operating system
     HOPE programming environment
     meta-language drives transformation system
     PROLOG compiler
```

4. DISCUSSION

Dr Watson asked for confirmation that construction of the machine relied on producing one chip many times. Mike Reeve said that this was so. He also pointed out that the chip is of a complexity close to the limit of what is possible with current SERC facilities. The find fall back position is to use AMD bit slice logic to emulate the chip.

Dr Watson then asked for more details of how the packet format maps onto the hardware structure. Mike Reeve explained this. There are no problems with associative matching because when an agent seeks a packet from the ring, the packet's identifier is known. The packet identifier field is in bits (16 or 24 on prototype) wide. The agreement list is restricted to 3 entries (allowing Cond); Currying is used to deal with more arguments. The status and function fields fit in one n bit field. The signal list on on a bit field, but further entries can be chained into extra packets. The reference count is an n bit field. Thus the packet size is 6n fields. 5n signal requests can be accounted per extension packet. Access to the packet pool is through an n bit date path, because frequently only one field needs to be extracted to do useful work. Each agent holds a table of function definitions, loaded from the packet pool when the function first occurs.

5. HARDWARE FACILITIES AT IC

Martin Cripps gave a presentation on facilities for hardware development at IC.

Martin joined IC in 1969 as a research student. He came to the department as 'the hardware man' and is responsible for hardware and data communicaions activities. During 1972-3 he was involved in a Hardware Laboratory project. The department has two excellent technicians, whose chief skills are woodworking and metal working. The department wanted a hardware laboratory that could be left open 24 hours per day and would not need a technician in attendance.

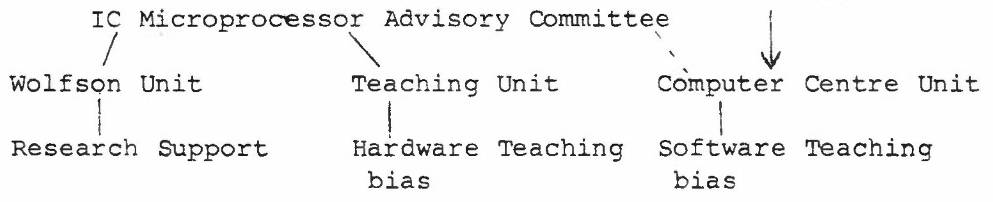
Martin conceived the idea of simulated hardware systems. The system runs on a Texas 960 mini to which are connected a series of units looking like logic boards. 16 boards can be run from one mini simulation from gates to registers to complete microprocessors.

Martin is now engaged in the construction of a cheap departmental terminal network called Sonet, which runs over 3 screened twisted pairs with crimp connections for each terminal. The network will support up to 128 terminals running at up to 19.2K baud (guaranteed).

An electronics Workshop has been built up, staffed by two ZRA grade people, one is a Chartered Engineer, the second is almost. A third may be taken on next year. The Workshop suggests hardware work in the Laboratory.

Martin is both a Senior Lecturer in the department and Director of the Wolfson Unit for Microprocessor Research.

Microprocessors are supported in several ways:



The Wolfson Unit supports any research that might be of use to industry and hence its resources would be available to the Alice project. The Unit has one and a half RA's (the half is a joint post with the Teching Unit). One has a PhD, the other is a graduate. The Unit tries to provide linked hardware and software testing environments. Emulation facilities are available for most types of microprocessor, using HP64000 series equipment. The Unit also has logic state analysis and one very high quality storage oscilloscope.

6. FURTHER DISCUSSION

Dr Watson asked how many boards they expected to have to construct.

The desk top Alice has 24 agents, each requiring a single card (double eurocard). The packet pool is between 4 and 8 cards. It is likely they will have to build about 40 boards, but many will be the same. They would like to use pcb construction but have no design facility in-house.

Dr Watson, speaking from experience with the Manchester Dataflow machine, strongly advised them to look to subcontracting pcb layout and construction. A number of companies now offer such services, and the investment is well worthwhile. John Darlington said they would be only too pleased to follow this advice.

Martin Cripps commented that he had considered renting a CAD system for 1 year with a view to purchase thereafter.

7. DEMONSTRATIONS

A very impressive set of demonstration was mounted. The HOPE compiler and Metalanguage Transformation System were seen in operation on the DEC10 with a variety of examples ranging from the binary factorial algorithm to Dijkstra's algorithm for calculating the digits of e (based on the fact that the i'th digit is 1 in base 1/i!). The source of the HOPE compiler in HOPE was also available for inspection and seemed extremely well written. Various examples of Alice CTL were displayed.

Mike Reeves' Alice simulator and statistics analysis suite were seen in operation on the departmental IBM 4341 machine. All other users were removed fromthe machine for the demonstration so that response would be reasonable. John Darlington and Mike Reeve use more computer time running the simulator than the rest of the department (staff and students) together. The binary factorial example and a number of other examples (including list doubling) were executed and simulation results were displayed. Mike Reeve seemed to have done a thoroughly professional job of work.

8. PRIVATE DISCUSSION

Dr Watson and Dr duce then had a private discussion to agree feedback to the main Panel meeting on 8 February.

1. Dr Darlington's group are <u>extremely</u> impressive. All are research workers of the highest calibre who have done a thoroughly professional design study to date.

2. The key person in the project from the hardware side is clearly Mike Reeve, who combines software and hardware talents to a rare degree.

3. The group would be well advised to contract out artwork and board manufacture. Dr Watson estimated this could be done for about £20-30K (rough estimate). This could save an enormous amount of time.

4. The one point of concern is that the project stands or falls on the ability to produce the chips. The risks have been reduced by refining the design to one chip, much of which is standard (ALU, RAM, i/o ports) and should be available in a library manufacture however remains a risk.

   Ways should be sought to reduce this risk, possibly by involving an industrial concern as a fall-back source.

   The group should also be encouraged to organise the design such that if chips cannot be manufactured, the machine can be built from conventional components.

5. Dr Watson was of the view that in spite of the risks, it is right to take as a startig hypothesis 'Alice can be implemented efficiently in VLSI'.