ICL 2900

THE NEW SX MODELS

Foreword

Having been involved in the development of ICL's Series 39 range throughout its entire history, it was my particular pleasure to be asked to introduce these three papers on SX Systems. These articles cover the development of the SX Systems node – which was designed by a core team of 120 increasing to 200 at the peak of its activity. However this number would be at least doubled if one were to take into account those who designed the accompanying Macrolan, VME developments, Node Support Computer, Macrolan/Oslan Gateway, and the Design Services infrastructure. The commitment of all these people was required to see the project through to completion.

I have been fortunate enough to lead the development of Series 39 SX Systems from their inception in 1986 right through to the present day. From the very beginning I was aware that the technological advances incorporated in this new product could return ICL to the forefront of mainframe design. In fact the resulting processor has now been announced as the most powerful uniprocessor in the world. The unique architecture of this processor forms the subject of the first paper, 'The SX Node Architecture'.

Behind this architecture lies a lengthy design process. As Director, Mainframe Systems it was my responsibility to provide the team of designers who worked on the Essex project with the best silicon technology possible. Accordingly, in 1985, we extended out collaboration with Fujitsu to include the Hawk technology which gave the ICL designers access to the most advanced chip technology in the world. However, we were aware from the start that our competitors would also have access to the same basic technology. Accordingly, it was up to our own designers to ensure that their processor and associated hardware designs exploited the new technology to its greatest advantage, and thereby gained true superiority for the SX Systems range. The second article – 'SX Design Processes' – describes how design objectives were set and met through the most advanced methodologies.

Their success can be measured by a comparison between SX Systems and a mainframe using the same Fujitsu technology manufactured by one of our competitors: Amdahl's 5990. The SX 580–20 is less than a sixth of the 5990/1100's weight. It takes up less than a quarter of the floor space. It uses less than half the power, and it puts out less than half the heat. And yet it gives over 50% more power per processing node. The story of how these particular advantages were gained forms the basis of the final article: 'Essex Packaging Concepts'.

While the lower end of the computer market has become increasingly commodity led, the superiority of Series 39 SX Systems is a clear demonstration that design excellence can produce true differentiation for a mainframe range. It is this last point which I hope that you will bear in mind as you read the following articles. ICL has long had the reputation for breaking new ground. By its pioneering adoption of such advances as optical fibre cabling, and its decision to stand out against older thinking by developing an operating system better suited to the information needs of the current marketplace, ICL has always endeavoured to create products which pass on unique benefits to our customers.

T.A. Hinchliffe
Director, Computer Products Division

The SX Node Architecture

J.R. Eaton, G. Allt and K. Hughes
Corporate Servers Product Group, ICL, West Gorton, Manchester, UK

Abstract

SX is the latest generation of high performance Series 39 processors. It is compatible with existing Series 39 systems but offers much higher performance both in single and multi-node configurations. SX also offers increased I/O connectivity and performance to balance the increase in processing power. This paper describes some of the more important features of the node.

1 Introduction

SX is the latest member of the successful Series 39 systems. It is introduced to provide increased levels of performance for users of the ICL VME operating system. As a member of an existing family it must be completely backwards compatible; all current user programs will be capable of running on SX without the need for recompilation.

Many design and technology innovations have been combined to give the SX the increase in performance it has achieved. In many respects this has only been possible because both the technology and the design capability came together at the right time.

SX is built from large ECL gate-arrays manufactured by Fujitsu. These are significantly larger and faster than those used in the Series 39 level 80. However the technology alone could not have delivered the required performance increase over the Series 39 L80. SX has relied heavily on design innovation which has lead to the development of a very sophisticated pipelined processor – possibly the ultimate pipeline for the 2900 order code. The result is a design which uses three times as much logic as the Series 39 L80, has a clock beat less than half that of the Series 39 L80 *but* has increased the performance by a factor of up to four. We have obtained a performance boost of two by technology and two by design.

In the early design of SX it was recognised that the main business of the users of VME based systems was to be Online Teleprocessing – OLTP. It was decided that this was one area where the node would have to be optimised but not at the expense of other workloads. A great deal of design effort –

particularly in the Order Code Processor (OCP) slave has been done to achieve this.

The Series 39 L80 was provided with optical fibre Input/Output, Macrolan 50, for high speed peripherals and OSLAN (Ethernet) for slow speed peripherals. For SX, in order not to compromise the design of the high speed I/O system, it was decided not to have slow connections directly to the node but to provide OSLAN connection via the Macrolan to OSLAN Gateway – MOG.

As part of the Series 39 family, SX must provide a multi-node capability. A huge amount of effort has been invested to provide a highly effective inter-node service which is technically capable of supporting a multi-node system with up to 8 nodes.

1.1 Existing System Behaviour

Extensive investigation into existing Series 39 L80 was carried out in order to understand the behaviour of the current systems. A processor was extensively modified in order to capture large amounts of trace data relating to instruction sequences and data addresses. Many monitoring runs were carried out using different workloads, from teleprocessing to scientific batch. This information gave the designers an understanding of the current use of the Series 39 systems. Together with awareness of the expected trends, this information allowed the optimisation of the SX design for the workloads which were actually being and going to be run. The trace data was used during the design and development process to predict performance for a number of workloads and used for exercising high level models in order to identify bottlenecks in proposed designs.

1.2 Outline Node Description

The SX node consists of four units

Order Code Processor	OCP
Input Output Processor	IOP
Inter-Node Processor	INP
Store	

Although these units are logically separate, much of the functionality of the INP and IOP is provided by the OCP.

The basic information flow – data and control can be represented by the following diagram:-

2 The SX Order Code Processor

The SX OCP is a heavily pipelined design. At the highest level it consists of four asynchronous units.

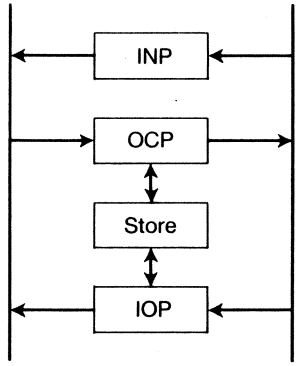


Fig. 1 Conceptual data and control flow in the SX node

Scheduler
Upper pipe – address generation
Data slave
Lower pipe

The Upper Pipe, Data Slave and Lower Pipe are often considered as a single unit – the engine.

2900 order code is fetched by the scheduler and then decoded to give a sequence of Picode which the engine then obeys. Picode is the implementation order code of the SX OCP – it takes the place of microcode on previous machines but is at a higher level, much closer to the 2900 order code. Picode is executed by the engine.

Each unit is internally pipelined with typically four stages. The internal stages of different units are effectively overlapped. This allows an output

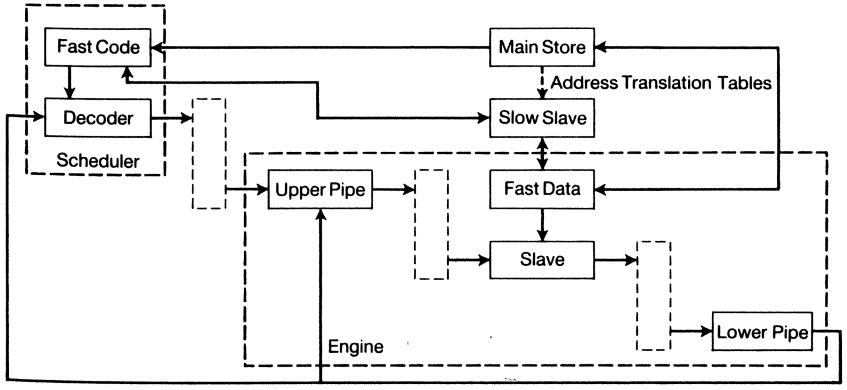


Fig. 2 The SX pipeline

from a unit to be generated before that unit reaches its last internal stage. In the engine there are a total of 13 stages but after accounting for stage overlap only 10 are visible due to the overlap between stages. In the total pipeline there are 17 stages with 14 visible.

Each of the four units operates asynchronously. There is a queue of up to 16 instructions between stages – the parameter files. This allows the pipeline utilisation to be maximised. Instructions are executed in strict chronological order – this is a rule imposed by the Series 39 architecture – and completed at a maximum rate of one per beat. Although completed in strict chronological order the slave will handle instructions in any valid order. This allows store accesses to overtake one another subject to strict rules and hence allows the possibility for two read instructions to be completed in reverse chronological order provided that the destination registers were different and that no conflicting writes appear in between. The lower pipe will ensure that the final terminations are in the chronological order.

The pipeline handles two separate streams, the A and the B stream. The A stream is used to execute 2900 order code, the B stream to provide system level support. There is communication between the two streams in order that certain activities may be co-ordinated. This is handled by two First In First Out buffers, an A to B and a B to A. These cause events to be generated on the target stream. The use of two streams has the effect of maximising the pipeline utilisation by absorbing the slack caused by store latency etc.

All registers are held and maintained by the lower pipe, for both streams – each stream has its own copies of the register set. Copies of certain registers are held in the upper pipe and the scheduler in order to improve performance by reducing the latency of access. The upper pipe holds copies of all registers which can be used in address generation and attempts to maintain them, but in certain circumstances it cannot. A prime example of this is when the register is loaded from store. In these cases the register will be updated from lower down the pipe. In general this update will come from the lower pipe; however in certain circumstances the slave will provide the data directly. This has the advantage of significantly reducing the latency for certain updates. This leads directly to a performance improvement for the OCP.

Like any other pipeline the SX pipeline needs to be reset after certain events such as an incorrect jump prediction. In this case the current instructions being executed by the pipe must be abandoned. This is handled by resetting the pipe and then updating any slaved copy of a register with the current master value from the lower pipe. This is an expensive event and the frequency must be minimised in order to maintain the performance of the pipe. This was achieved by developing a very effective jump prediction mechanism.

Interlocks between units are handled using a scoreboard-like technique. This is intentionally "imprecise" in that instructions in any unit will start

regardless of the state of the scoreboard. At the end the scoreboard is used to determine if that instruction has been completed successfully. This allows many of the bypasses between stages – where data is fed back up the pipe – to be used. The effect is a net gain to the performance of the pipe.

2.1 Scheduler

The scheduler is responsible for fetching the 2900 instructions (PLI = Primitive Level Interface) and for translating them into sequences of Picode which will be executed by the engine.

PLI fetches are satisfied either by the fast code slave, the slow slave or by the store. Picode fetches are satisfied by the Picode RAMS (a special case) and also by the fast slave, slow slave or main store.

A sophisticated jump prediction mechanism is implemented for the 2900 order code. It is essential in order to maximise the performance of the pipe. Without this feature, the number of jumps in normal instruction streams would disrupt the pipeline and reduce the performance dramatically. Various jump prediction mechanisms have been investigated in the past giving various results depending on the workload. The method chosen for 2900 order code on SX is to access an indicator, using a hash of the jump instruction address, which predicts if the jump will be taken. The indicator is set based on the behaviour of the jump instruction in the past. A simple by 1K by 1-bit RAM is used to hold the jump prediction table. This gives a jump prediction accuracy of over 85% on typical workloads.

Picode has a different jump prediction mechanism. All Picode jumps contain an indicator as to whether the jump is likely to be taken or not. In the design of the Picode it was easy for the programmer to determine which way the jump was likely to go, setting the indicator accordingly. This has resulted in very high prediction accuracy for Picode – in excess of 95%.

2.2 Upper pipe

The upper pipe is responsible for address generation and for the execution of simple instructions.

Address generation for Picode, which maps very closely to 2900 PLI, is a relatively simple task BUT is complicated by the architectural rules associated with 2900. The upper pipe handles all these cases. It will check stack addresses to ensure that they do not fall outside the stack frame and will check that code area addresses are within the current code area.

Some simple instructions are executed directly by the upper pipe. These are restricted to simple register/literal operations or register/register operations. This has a significant effect on the performance of the pipeline by reducing the effect of pipeline stalls introduced by register dependencies. If the unit did

not execute the instructions directly, the latency of a register update would be the complete length of the engine pipe; this would clearly be unacceptable.

2.3 Slave

The slave is essential to the OCP design as it provides fast access for both code and for data. A multi-level slaving system is provided with fast code and data slaves supported by a slow slave and finally the main store. I/O is always direct to main store. A logical view of the slaving system is shown in Fig. 3.

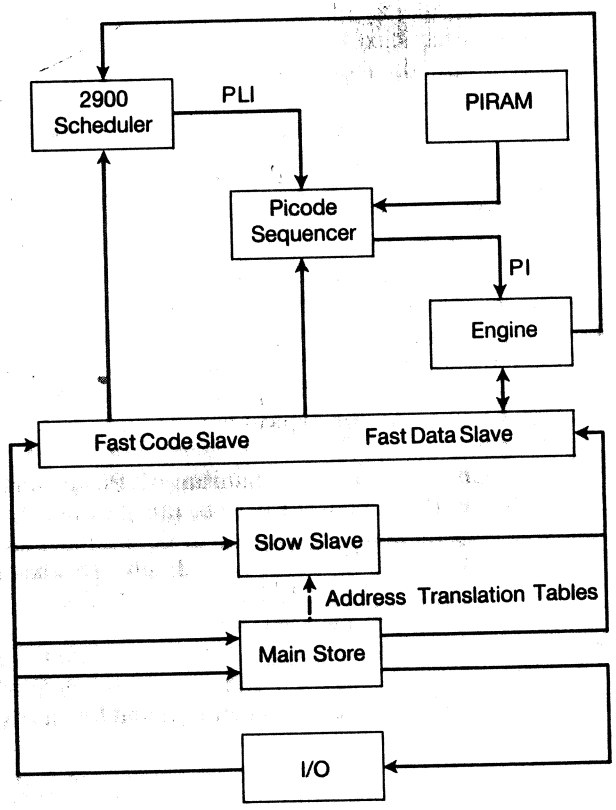


Fig. 3 The SX Slave Hierarchy

The slave handles access to all operands, those in virtual store, real store, image store (control areas) and working store. Even operands which do not require any slave access, such as literals, are passed through the slave. Image store and working store are special areas within the fast data slave. Image store is used for machine control; typically it is used for communication between the independent units of the machine such as the I/O system.

Working store is special store dedicated to the Picode and will typically hold architectural and process control information.

2.3.1 Slave hierarchy The slave has a number of specific slaves used for different purposes which together can be viewed as a hierarchy.

fast code slave	128 lines of 32 bytes	– fully associative
fast data slave	32 lines of 32 bytes	– fully associative
slow slave	64Kbytes, 32 byte cells, 4 way semi-associative	

The fast code slave exists within the scheduler. It is used to hold code for both 2900 PLI and for Picode. It is directly in the internal pipe of the scheduler and as such costs one pipeline step to access.

The fast data slave exists within the engine. It holds all data read from the store. Associated with each line of the fast slave is a set of byte-valid markers. These indicate the validity of the bytes within the cells and are used to eliminate the need for store access on cell creation. Access to the fast slave is in the engine pipeline and as such costs 1 beat to access, which is part of the pipeline delay. The fast data slave is a delayed write-through slave. Only on an explicit prompt or when the cell needs to be re-used is the cell written to the slow slave and/or store. This has the effect of greatly reducing the store bandwidth required for write access; the fast data slave behaves as 32 store buffers.

For the fast code and the fast data slave when a cell is to be re-used the current data is unloaded into the slow slave. This is the only route by which data can get into the slow slave (except for address translation tables – see below).

The slow slave supports both the fast code and data slaves. It is split into four parts of 16Kbytes each, one holding code, one holding data and two holding address translation tables. The slow slave is probably mis-named, the penalty for access to the slow slave being only 2 beats, slower than the fast slave but as fast as (most) slaves in previous machines.

Slaving the address translation tables is an innovation on SX. Previous machines have never slaved address translation tables, only "Current Page Registers" (CPRs) which are manufactured from the address translation tables. Slaving the address translation tables provides fast access to the translation data which results in fast creation of the address translation slave entries. This is important as it allows the number of address translation slaves (CPRs) to be relatively small.

2.3.3 Slave support of address translation A number of Current Page Registers (CPRs) are held for both code and data. These map the virtual address to the real address and provide information on the protection levels of the page. In addition separate CPRs (global CPRs) are used to map virtual

addresses which are indirect virtual addresses which map to a global virtual address. The CPRs are fully associative with a least recently used (LRU) replacement algorithm. The CPRs are allocated as follows:

Code	6 Global
	6 direct
Data	6 Global
	12 direct

The address translation process relies on both the global and direct CPRs. A virtual address is first translated using the global CPR. If the address is indirect then the global CPR recognises it and replaces it with the global virtual address. The address is then presented to the direct CPR which will generate the real address and all the protection information required. Both the global and direct CPRs are directly accessible in the slave pipeline. In this way the cost of global translation is eliminated. On previous machines the cost of indirect address translation has been very high, up to 50% greater than direct address translation. Global areas are used to share data between virtual machines which is a facility heavily used in OLTP systems. The substantial improvement in the cost of address translation has the effect of significantly improving the performance of TP systems without impacting other workloads.

2.3.3 Slave Control Interface On SX the operating system has a procedural interface to the OCP which is used to manipulate the address translation tables. The information passed is used by the OCP to determine what action must be taken in order to maintain the slave. This is another example of the tendency to offer higher level interfaces to the operating system in order to insulate it from changes in the underlying hardware. The effect on SX is to minimise the cost of slave clearances leading to increased slave retention. This has a direct influence on the machine performance.

2.3.4 Data alignment The slave is responsible for operand alignment for both read and write data. When store data is read – whether it is supplied by the fast slave, slow slave or main store it is aligned before being presented to the lower pipe. Similarly when data is being written it will be byte-aligned before being written into the slave cells. This gives the machine access from 0 to 8 bytes on any boundary, including page straddles. It is the first machine developed by ICL which is capable of doing this and makes SX the first truly byte oriented VME machine. Byte oriented access simplifies the Picode in that it never has to handle complex cases explicity such as when a six-byte field is contained in three words. It also simplifies the validation process for the Picode as the special cases do not have to be tested in every sequence.

2.3.5 Execution sequence The slave is not constrained to operating on its input request stream in a chronological order. The slave will action VALID requests in the "oldest first" order. This has the effect of allowing the slave to do operations which are not in sequence. The slave continues to fetch data

and often will fetch data ahead of when the lower pipe will require it. This has the effect of improving the overall pipe performance.

2.3.6 RAM protection Hamming correction is provided on all data areas for improved reliability and integrity. This is an improvement over previous designs where Hamming correction was provided on the main store but only parity error detection on other data stores.

2.4 Lower pipe

The lower pipe includes the main execute mill. It supports functions for all the Series 39 data formats: integer, decimal, floating point and logical.

The lower pipe provides complex functions such as floating point multiply and divide. These are micro-sequenced using a soft control program held in very fast RAM – the execute mill microcode. The lower pipe operates on a 64 bit data flow. This improves performance as 2900 uses 32-, 64- and 128-bit operations although 128 is infrequent.

In addition to the mill function the lower pipe updates the registers from a mill operation. This will result in copies of that register held in the earlier stages on the pipe (typically the upper pipe) being updated.

Only instructions which successfully terminate can result in register updates. Instructions which terminate unsuccessfully – Virtual Store Interrupt, Program Error etc. cause an exception which results in the appropriate exception handling code being run.

The sequence of execution of the lower pipe is strictly chronological. Although the proceeding stage (the slave) executes instructions in any valid order, the instructions must be terminated chronologically. If this was not the case, restart after a virtual store interrupt etc. would be extremely difficult!

3 Picode

Picode is the native order code of the SX machine. It is fetched by the scheduler and processed by the upper pipe/slave/lower pipe. Picode has been designed explicitly for the SX machine which is a 2900 execution machine. As such many of the 2900 concepts, such as descriptors, are built in.

Picode is unlike the previous low level order codes on Series 39 machines in that it is not a microcode. Picode is at a much higher level than microcode but slightly lower than the 2900 order code. Unlike microcode there are no timing rules associated with Picode. Each instruction executed is independent: interactions are only through the operands addressed by the intstruction. This is enforced by the hardware. Picode execution is atomic in that it is either completed or not, it cannot be interrupted part way through.

Picode has access to a larger register set than the 2900 order code. Nominally the 2900 register set is duplicated BUT the individual 32-bit components of larger registers are accessible. Each instruction operates on two operands, one of which can be in store. Both are used as source operands and one is used as the destination.

Picode is held in virtual store and is accessed through slaves in the scheduler. Three levels of slave exist for Picode, a very fast Picode RAM, the fast code slave and the slow slave. Separate Picode stores are used for the A and B streams with independent Picode RAMs for each. Each Picode RAM holds 1K instructions which correspond to the first 1K instructions of the Picode in store. The Picode RAM holds performance critical code – code which if held in the slave would be "thrashed out" by other less critical code.

PLI Execution is handled by translating the PLI into a Picode Sequence which is conceptually fetched from store and fed down the pipe. Since Picode is slaved the cost of fetching it from store is virtually eliminated. The basic Series 39 PLI is implemented in a very small amount of Picode – about 1K, the complete Picode for all PLI is less than 4K. This is achieved by providing a mechanism in the Scheduler which maps the Series 39 PLI onto a Picode template. Functions, registers and operands are replaced by changing the template into the real Picode instruction. The majority of executed PLI will map directly onto 1 Picode.

The Picode and hardware combine to optimise for the normal case. The normal case is executed as fast as possible. In order to support all other cases a special facility is provided called "warp". This causes the abnormal case to generate an exception event which causes entry into a special Picode sequence which will then perform the required action. "Warps" are vectored in that a specific sequence is provided for each. The "warp" behaves like an escape sequence in that at the end the originating sequence is returned to. In many cases the "warp" is generated by the slave as a result of an operand access. The operand access is converted to an access to the special working store where a picode address is found. This is then used to force a jump to the appropriate sequence.

Optimising for the normal case is also applied to the detection of program errors. In order to improve performance of the PLI implementation certain architectural checks which could result in a program error are handled imprecisely. Effectively the check is applied BUT the resolution is only to the instruction. A special Picode module – Resolution of Program Errors (ROPE) – is entered which emulates the instruction in order to determine the error precisely.

In addition to PLI support, Picode is used for all system level functions, which include

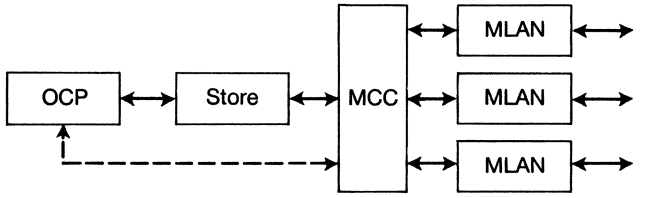
Multi-node support
IOP support
I/O protocol support (VME sub-system)
Node support (diagnostics etc.)
Etc.

This accounts for the majority of the Picode. In excess of 40K instructions are used to support the system level functionality. Such a large amount of storage would have been prohibitively expensive and could not have been implemented within the constraints imposed by the technology. Holding the Picode in main store and relying on very good multi-level slaving has allowed the use of large amounts of control function to be implemented in Picode without significantly degrading system performance.

4 Input/Output system

The input/output system on SX provides up to 16 Macrolan channels. These are used for all peripheral access to both fast and slow devices.

The I/O hardware is made up of a single Macrolan Channel Concentrator and up to 16 macrolan daughter boards.



MCC=Macrolan Channel Concentrator

Fig. 4 The SX I/O Interfaces

High speed I/O is driven using the same VME interface as was used on previous Series 39 machines – this has removed the need to change VME for all disc and tape driving.

For low speed devices the OSLAN protocol is handled over the Macrolan. A new subsystem has been introduced to VME – Remote Coupler Handling – to support this. This packages up the OSLAN data and passes it over the Macrolan to the Macrolan to Oslan Gateway (MOG) which then drives the OSLAN.

As on previous Series 39 machines the OCP supports the VME operating system by providing high level functionality for I/O driving. Logically the I/O driving software exists as part of VME; however the performance critical paths of the I/O interface have been implemented in Picode and are called

directly (and indirectly on interrupts) from VME. On SX this code has been extended to support the Remote Coupler Handling software.

The OCP supports the I/O by use of special stream B Picode. This is assisted by the use of special hardware entry points which direct the Picode to the appropriate sequence.

5 Multi-node

Multi-node systems are an important part of Series 39 strategy and hence essential for the SX system.

Series 39 multi-node systems are built around a number of nodes running a single coherent copy of the VME operating system. These nodes may be physically separated by up to 2 kilometres. Nodes may share virtual store by replicating that store on the different nodes. The replicated virtual store is maintained by the inter-node service without the intervention (in normal operation) of the operating system.

Experience of past machines has indicated that the performance of the inter-node service is highly dependent on the amount and type of inter-node traffic. In order to ensure that the SX inter-node service could handle all levels of workload, extensive simulation, using trace data from the L80, was carried out.

All previous implementations of the inter-node service have been significantly different, resulting in different VME sub-systems for the control of each. SX has introduced a new high level procedural interface for inter-node service control allowing VME to be insulated from the implementation of the inter-node service. It is now possible to replace completely the inter-node service below the VME interface without changing VME. This is a direction more of the VME interfaces will take in the future giving greater flexibility to the hardware designers while reducing the demands on VME to "box follow".

Due to the high performance of the individual nodes in an SX multi-node system, a high performance inter-node service is required. This is because an increase in node performance increases the amount of multi-node traffic. Handling this becomes a factor limiting the overall system performance. Nodes may be separated by up to 2 kilometres but this has an impact on the multi-node performance due to the message latency, primarily for synchronisation events.

The SX inter-node service is based on multiple glass fibre links – as have all previous services. Four optical fibres may be used as data links and one for chronology. For resilience and system partitioning purposes a second token ring is provided.

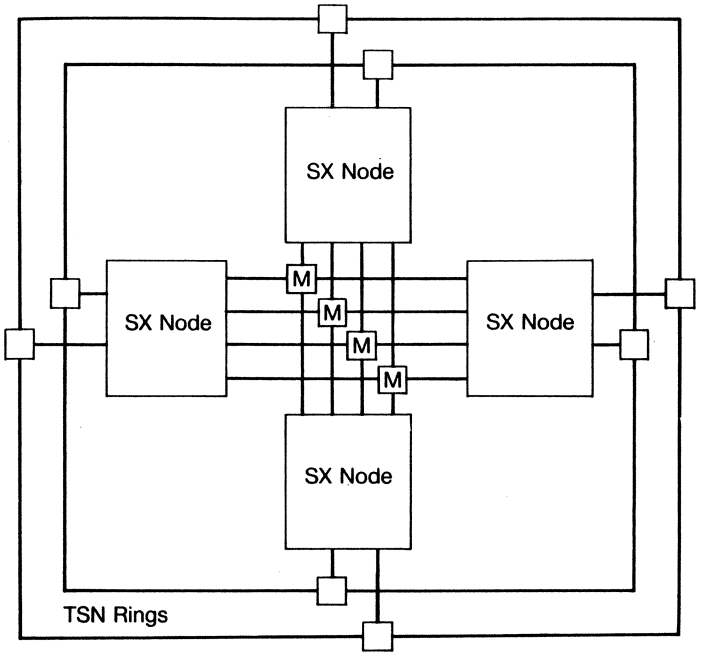


Fig. 5 The SX Multi-node Interface

This is a Macrolan Port Switch Unit (MPSU) which connects the Lan into a 'ring structure'.

All Macrolans use the same optical fibre technology-Macrolan 200.

Note: only one TSN ring is active in a four node system. When the nodes are partitioned one TSN ring is allocated to each partition together with a number of data LANs.

It is a requirement of the Series 39 multi-node system that a coherent store image is maintained. On the current multi-node systems, store can be replicated on different nodes, and this replicated store image must also be maintained coherent. This is achieved by applying a strict chronology to all writes to this store. On SX the chronology is provided by means of a Token Serial Number – TSN – which is carried by the single token on the TSN ring.

A node wishing to transmit a public message waits until the token arrives on the token network. In addition to a serial number the token provides information on which of the data LANs are currently in use. Providing a LAN is free the node will acquire dedicated access to a data LAN and indicate this in the token. The token is not stopped; it is changed as it passes through the node. The node now transmits the message on the data LAN has acquired. Nodes which receive the message all "acknowledge" the message.

Once all nodes have acknowledged the message – a protocol within the data LAN – the message is deemed complete. The OCP is then informed. The sending node clears the information indicating the LAN is in use when the token next passes.

By separating the implementation of chronology and of data transmission, the latency for messages is much reduced. The circulation time of the token is fixed; it is independent of the traffic on the data LANs. This reduces the latency for semaphore operations which are *critical* to the performance of the system.

SX has introduced blocking to the transmission of data on the inter-node network. All outstanding messages are transmitted when a data LAN is acquired as a single "block". In a similar way multi-node traffic as a result of moves to public areas is "blocked". A string order which writes to an area which would result in inter-node traffic has the writes buffered. Only when this buffer is full or when the string order completes is the data actually transmitted. This significantly reduces the data LAN utilisation with corresponding improvements in performance.

As indicated above all replicated store must be maintained coherent across all nodes. It is possible that a write from one node clashes with a write from another – the same store area is being written concurrently from two processes. To avoid this resulting in different values in the stores a clash map is implemented in each node. If a write from another node addresses the same store area before a write from the current node is completed, ie before the OCP is informed that all nodes have acknowledged the message – then a clash has occurred. If a clash has occurred then the nodes own writes are re-applied as they return. The clash map – implemented in the slave – detects the condition and inhibits all reads from this area, otherwise the OCP would have to be stopped until its outstanding writes were completed. This has the effect of maintaining performance as the resolution on clashing addresses is so high that clashes rarely occur.

6 Conclusion

SX is the most complex and powerful machine ever built by ICL. It has combined technology provided by Fujitsu with design skill and experience from ICL. SX has been a challenge to the designers and implementors. Many new and complex features have been introduced into the node, features which will be exploited in future designs.

The result is a high performance, highly reliable system capable of satisfying a large range of customers' computing needs.

References

1 BROUGHTON, P.: Input/output controller and local area networks of the ICL Series 39 Level 30 System, 1985 ICL Tech. J *4* (3), 260–269.

2 WARBOYS, B.C.: VME nodal architecture: a model for the realisation of a distributed system concept, 1985, ICL Tech. J. *4* (3), 236–247.

European (EP) and British (GB) Patent applications relating to SX

EP 262 782
Multi-node data processing system
D. Bell

EP 352 935
Data processing apparatus
J.R. Eaton, P.V. Rose

EP 357 188
Pipelined processor
J.R. Eaton, C.M. Duxbury

GB 2 215 887
Data memory system
J.R. Eaton, R.M. Lister, K. Turley

EP 320 098
Jump prediction
S. Hodges

EP 372 751
Pipelined data processing apparatus
C.M. Duxbury, P.V. Rose

GB 2 227 584
Data processing system
D. Bell, C. Lomas

SX Design Processes

G.P. Abraham, D.C. Freeth and H. Vosper
Future Products Development, Corporate Servers Product Group, ICL Computer Products Division, West Gorton, Manchester, UK

Abstract

This paper gives an overview of the design processes followed during the design of the SX system.

It covers some of the major initiatives and innovations and specifically highlights the extensive methods employed in design verification.

Tools described include performance models for OCP, I/O and internode, software emulation techniques, hardware simulation and test software.

The paper also contains a description of the design control and measurement methods.

1 Introduction

The continued improvements in silicon technology and the increases in complexity have necessitated that the design of the SX system uses design methodologies which are innovative, rigorous and give orders of magnitude improvement in the quality of the design compared with previous processor designs.

The methodologies and associated development routes described here cover the design of the hardware, the associated microcode known as PICODE and the node support computer (NSX).

There are many examples within these areas where a high level of innovation can be shown and this paper focusses on some which although particular to SX may be of interest and applicability to many areas of system design.

All of the initiatives are a direct result of analysing the problems encountered on previous projects and looking at areas for improvement whilst at the same time taking into account the advances made in the hardware and software tools that are now available.

To set the context for the following sections, reference to a generic development route is helpful – see Fig. 1. This 'V' diagram illustrates the stages of design from system down to implementation on the left hand side and stages of test from unit up to system on the right.

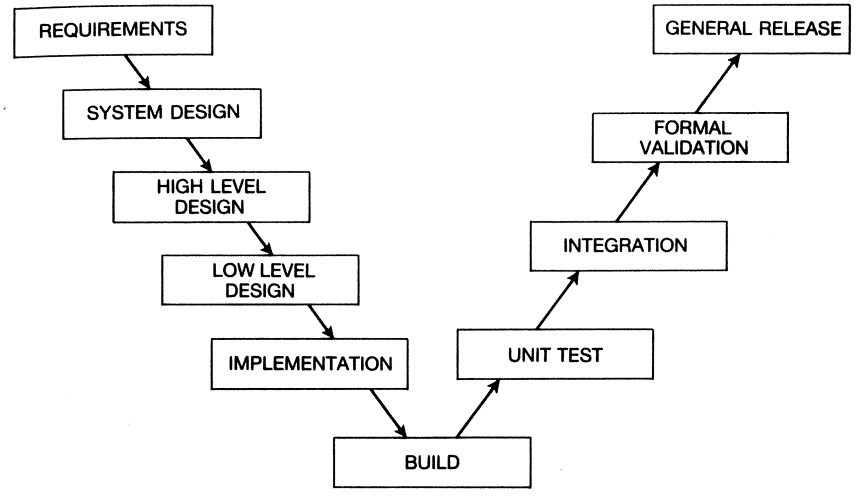


Fig. 1 Generic development route

The aspects of this development route which will be covered are the design and development activities, i.e. from Requirements through to Implementation. Particular emphasis has been placed on the System Verification, Simulation and Evaluation and Test Bed activities which have demonstrated considerable success in the System, Picode and Hardware Design processes.

2 Requirements/system design

Fig. 2 shows an outline of the overall design and validation process in terms of the design data decomposition and the verification method for each level. For example, the logic description is validated by the use of the MSIM logic simulator.

This section will describe the requirement setting process and two of the most innovative and valuable verification techniques used at the System Design stage. These are:-

HAM – High Level Architecture Model.

and

SIMULA – A model of I/O and multi node capability.

The other design validation methods will be described in Sections 3, 4 and 5.

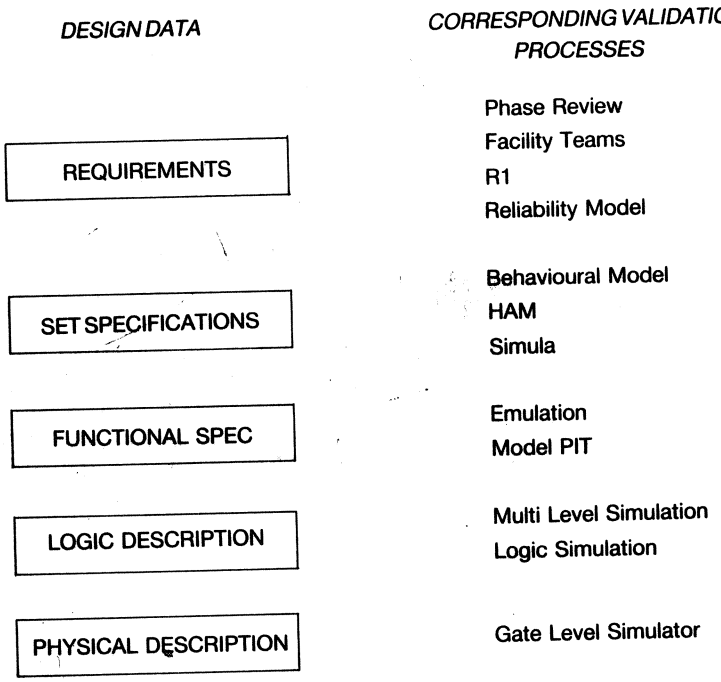


Fig. 2 Overall design and validation process

2.1 Requirements

A top level Statement of Marketing Requirements is formed reflecting the customer requirements. This is mapped to a System Definition taking into account technology information and the Business Case. Having established an issued System Definition containing targets for the various system attributes, the mapping and decomposition of these requirements is an important process.

In the case of SX, system attributes were managed by setting up orthogonal review mechanisms which were led by Systems Architecture and drew together expert areas of the Division. Thus, attributes such as performance, reliability and maintainability could be monitored and assessed. Component and system reliability models were constructed and updated with lower level design data at appropriate times during the development life cycle.

Performance models will be discussed in more detail later.

Facility teams were also created within the Development teams and a three level Requirements mapping and review mechanism established. These multi disciplinary teams each investigated specific areas of the design and agreed the definitive set of requirements for that area. These included, for example,

Memory, Order Code Processor, Diagnostic Control and Engineers Facilities. These high level requirements were then reviewed at three stages:-

The definition of *what* facilities were to be implemented (R0), *how* they were to be implemented (R1) and then the *mapping* of these facilities to functional partitions of the design (R2). The completion of this stage allowed detailed design implementation and verification to proceed.

2.2 HAM

The High Level Architectural Model – HAM (patent pending) is a model used to predict the performance of the SX system. It is a behavioural synthesiser – that is to say, it does not execute operations. It builds up a behavioural picture of the node by using information derived directly from a Level 80 which is monitored whilst running standard customer benchmarks. This is combined with architectural functional patterns extracted from the design state of the SX node.

HAM is a model of the pipeline and slaves in an SX node which simulates their behaviour and interaction.

This modelling is achieved by decomposing the 2900 PLI [PLI = Primitive Level Instruction – the range compatible order code for Series 39] into individual pipeline relationships which are used to determine the behaviour of the model. These relationships identify the functional interactions which enable designers to calculate the performance costs of pipeline hold ups.

Virtual addressing information is also supplied to the model which permits slave algorithms to be assessed.

This meant that both "what if" scenarios in the design and changes resulting from practical design implementation could be modelled, giving accurate predictions of system performance.

One of the major advantages of HAM is its speed of operation which is one thousand times faster than conventional simulators. The resultant power advantage enabled realistic benchmarks and workloads to be run.

Significant effort was put into the extraction of real time paths through the running of benchmarks on Series 39 to capture instruction streams and addresses. This ensured that the model was not working on a statistical basis but on real PLI streams. The major benefit of this was that the pipeline operation and slaving were verified and established early in the design life cycle.

The model was used throughout the SX Development to monitor system and workload trends and give early and accurate performance assessments which were fed back to the corporate review process and confirmed that the project was on track to meet customer requirements.

The design of the I/O and inter-node subsystems was modelled using a variant of the Simula [a derivative of ALGOL] programming language. The system, Discrete Event Modelling in Simula was derived from a development by G. Birtwistle in 1970 [Ref 1].

Simula is a behavioural model which predicts I/O and Internode performance using standard queueing theory and statistical methods.

During the early design stages, basic requirements must be thoroughly understood. For I/O these requirements are usually specified as throughput rate and for inter-node as the performance degradation seen as a result of adding more processing nodes.

The I/O subsystem was modelled with particular emphasis placed on buffer sizes and processor loading. The bulk of I/O traffic consists of packets of data flowing between the node and the disk subsystems. Analysis was made of standard benchmarks to understand traffic profiles and provide parameters for the Simula models. The model was run in varying I/O configurations and a check made for any non-linear behaviour, for example data buffer overflow on peak traffic loads.

The I/O model was run with various limit conditions to ensure that the design was stable, could meet requirements and proceed to the next design stage.

The inter-node system model was developed to enable performance predictions for various multi node configurations and differing physical separations between nodes. As with the I/O model, behavioural information from various benchmarks was analysed to establish statistical patterns of information, in this case internode traffic.

In a multi node system, one of the prime causes of performance degradation is the loss of processing time when a node needs to resynchronise its local memory with that of other nodes. Synchronisation is achieved in 2900 architecture with PLI semaphore instructions. The distribution of these semaphores, together with node separations, was modelled to predict total system performance.

The model runs on a Series 39 mainframe and requires relatively small amounts of processing power. Graphical input is now available on SUN workstations which makes the Simula system an extremely flexible and useful tool in the System Design environment.

3 PICODE design process

One of the most notable process improvements and successful design implementations in the development of SX was in the area of PICODE design. PICODE, which is the native order code of the SX machine, is a derivative of Series 39 Level 80 microcode.

The delivered quality of the PICODE shows a twenty times improvement over its Level 80 counterpart in terms of faults found after prototype switch on. The two most significant contributors to this improvement were the design and implementation of a disciplined High Level Design process and an innovative module validation process, the Picode Test Bed (PIT).

3.1 High level design process

Fig 3 shows a simplified diagram of the overall PICODE design process. The High Level Design Process takes as its starting point the definition of the primitive level interface and the requirements from the R1 stage. PICODE specification documents are then generated for specific modules and each module is then flowcharted. PICODE source is then written and the source files are then linked into hierarchic structures and directly to their equivalent flow chart boxes using the SX design database. [Section 6].

It should be noted that one of the most fundamental aspects of the process is that design reviews, or desk checks where appropriate, are held after each stage. So for each specification, flow chart or source module generated, a formal review was carried out.

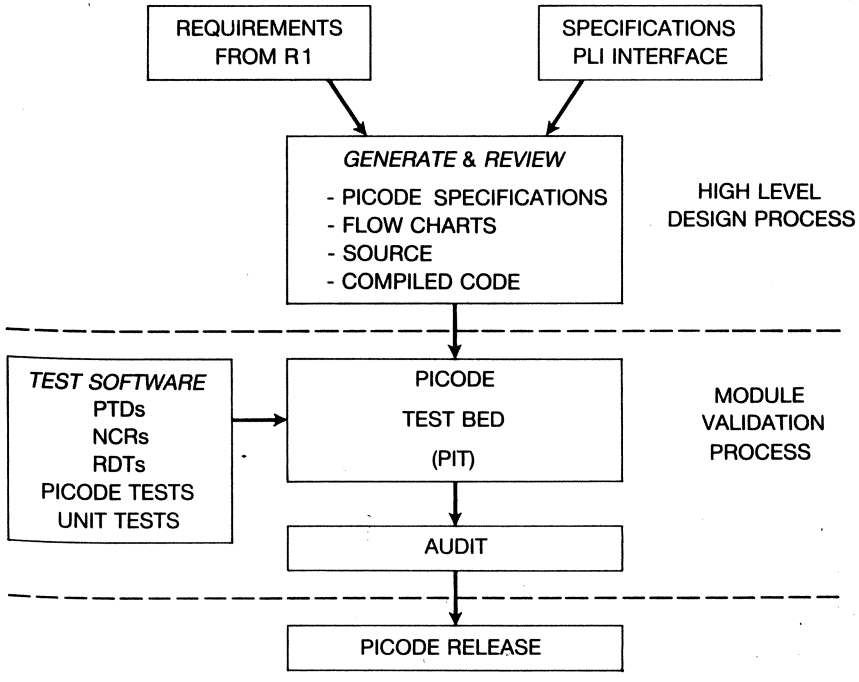


Fig. 3 PICODE development process

This review was fully defined in terms of procedure, input and output criteria and was performed and recorded by an independent member of the team.

Design control was performed via the SX Design Database which provided comprehensive build and control facilities. This precluded erroneous release or progress to next stage, and gave up-to-date and accurate audit trail information.

This highly disciplined approach meant that a very large proportion of faults were removed early.

A fault feedback mechanism was designed into the development process and fault information was centrally logged and analysed against targets for each design stage. A significant achievement was that 66% of all faults were found prior to the Module Validation stage.

3.2 Module validation process

Module validation on PICODE uses a particularly innovative environment – the PICODE Testbed or PIT, which allowed validation of the PICODE totally divorced from any hardware environment.

PIT is an emulator which permits PICODE to be tested, using specific and range defined standard tests. It provides a facility for testing sequences, executing PICODE and tracing execution paths. The definition of the hardware is provided by the hardware teams in the form of PICODE Procedures (PIPROCS) which are captured in the PIT environment. Compiled PICODE is input to PIT and then executed. One further facility is the ability to write test scripts to enable sequences to be initialised and executed in a controlled manner.

Tracing facilities are an integral part of the system.

The test software suite supported by PIT includes – PLI definition tests, Normal Commissioning Routines (NCRs) and Range Defined Tests (RDTs). These latter two test suites are identical to those run on Series 39 and SX prototypes, therefore maximising test cover and ensuring range compatibility.

All outputs from the PIT runs were audited prior to build and release. Design control was maintained by marking the correct status in the SX design database to enable the build stage to commence.

The prime benefit of PIT is that through earlier exposure to range defined test software, the faults are removed which would otherwise have been found on prototype hardware. In addition, early bug free and functionally proven PICODE (and test software) was available to the hardware simulation environment (MSIM) for hardware design verification.

4 Node support processor design process

4.1 High level/low level design

The Node Support Processor for SX has exploited structured design methods based on Ward & Mellor [Ref 2], supported by CASE (Computer Aided Software Engineering) tools. The software package is the design capture product EXCELERATOR/RTS and the supporting hardware a network of ICL professional work stations (DRS M60/M80), with links to dedicated VME service systems.

Adopting this approach, has provided a formalised design method that enforces a common way of expressing a design – a common language. This leads to improved communication within the project, consistency in the design and significant improvement to the review/audit process.

EXCELERATOR/RTS provides automatic checking of the validity of the design and improved ease of maintenance – any changes to one area can immediately be checked for impacts to other areas of the design.

Designs can be expressed through a combination of Context Diagrams, Transformation Graphs, State Transition Diagrams and Entity Relationships, all of which can be checked for consistency with each other at the various levels of definition.

The context diagram
This shows the external interfaces to the system at a high level and is used to ensure there is agreement on the original requirements.

The transformation graph
This considers what processes are needed to handle data and control flows. The flows of data are shown as solid lines and the flow of control as dotted lines. This is the first level of decomposition and will show a number of processes.

The state transition diagram
This allows the same set of processes to be represented in a different way showing how the system moves from one state to another.

The entity relationship
All the information captured in these diagrams is organised into entity types that represent the basic components of the system and many instances of an entity type will exist. Each instance is called an entity, and with large and complex systems it is easy to lose track of how these entities relate to each other. This diagram captures these relationships.

Figure 4 is an example of a context diagram for one module of the application firmware, in this case a top level representation of the application

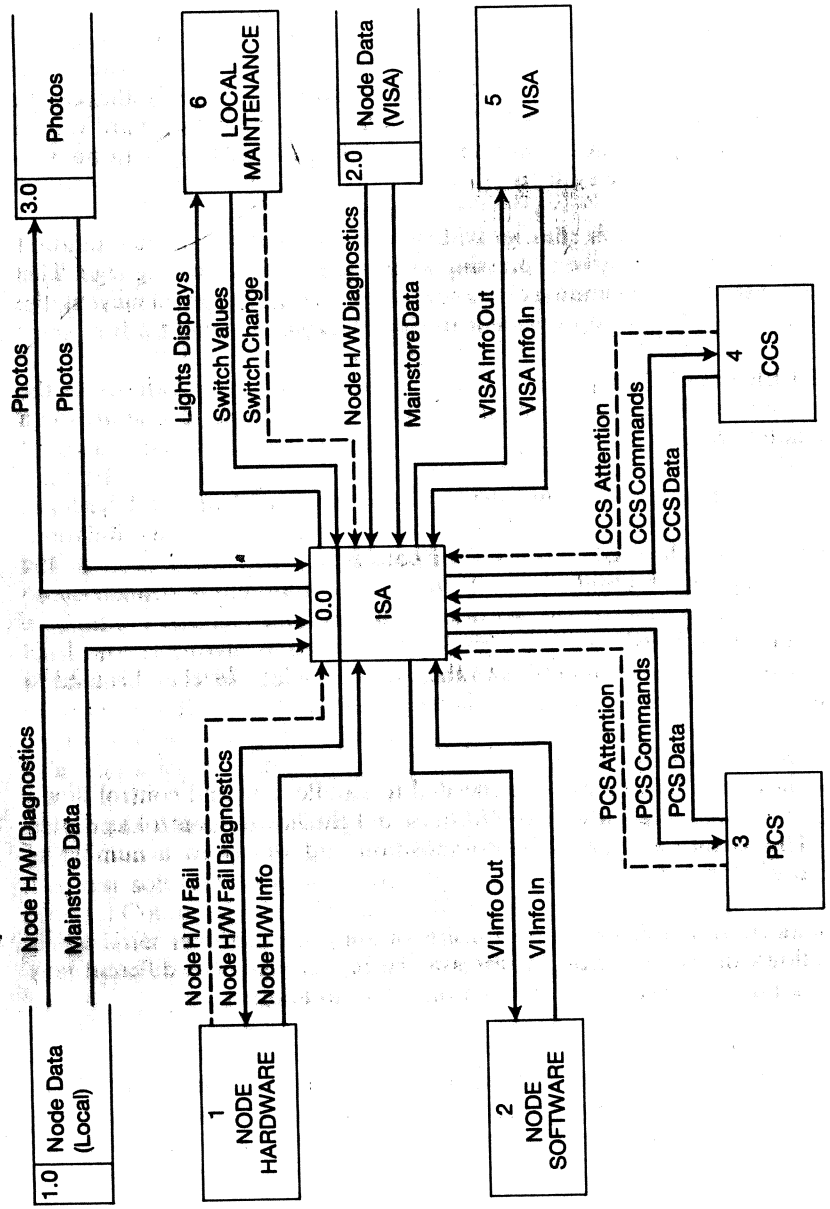


Fig. 4 Context diagram — application firmware (ISA)

firmware. It is typical in terms of showing a mixture of data and control flow and the level of information conveyed by such diagrams.

As part of the assessment of the technique, comparisons were made taking the original 'text only' design information for a particular module of design and the equivalent 'structured' design information. Critical analysis showed two thirds of the text was superfluous 'padding' that was not helping the understanding and the same level of understanding could be communicated in a few simple structured diagrams.

The benefits from the use of structured design methods and the particular implementation strategy adopted went well beyond the expectations of the team and showed quite clearly that it was a more cost effective process. A further benefit was the prevention of errors at the design stage, rather than trying to remove them in the later and far more costly integration and testing phases. This was borne out by the results of measures put in place across the complete development route.

4.2 NSX firmware test bed

The test bed phase of firmware development has proved to be a major area of improvement to the development route. It has allowed extraction of a large proportion of the residual faults post compilation of the code. This is in line with the fault removal targets of 40% in desk checking and 40% in Test Bedding, leaving only 20% to be found on test rigs or prototypes prior to full scale SX system integration. This compares with previous developments where only 25% of the faults were extracted in the Test Bed phase.

The NSX incorporates an INTEL 286 microprocessor that runs PL/M-86 programs. The project already had in place a network of DRS M60 workstations which allowed the PL/M-86 code to be tested in a proven test environment before being exposed to the NSX hardware.

However, it was noted that there had been no method of analysing the extent of test cover provided by the test bed, so the development team searched for software tools that could be used to do this.

A PL/M-86 version of the LDRA TESTBED software package was selected and installed. This provided not just a check of the testing level of the code modules, but also checks on the structure and use of non-standard code. As part of the assessment of the tool, existing Series 39 firmware was analysed and this revealed that the test cover had averaged only 60% – showing considerable room for improvement.

The LDRA TESTBED is a software testing and assessment product with built in quality management facilities for providing report and summary information, and it can run on most operating systems; in our case, VME.

Two types of code analysis can be carried out – Static and Dynamic.

STATIC analysis looks at three areas – *Standards*, *Complexity* and *Structure* and Fig. 5 shows a typical set of results from a run on some modules of the SSP firmware.

Module Name	Version No.	Version Lines	Executable Lines	Procedures	Standards	Complexity	Structured
CCST	1.18	2225	1280	10	4%	5	Yes
COMJOB	1.05	1586	648	38	4%	0	Yes
COMMON	1.66	2989	1227	35	5%	2	No (1
COMNMU	1.24	2176	789	19	1%	2	Yes
CPLRT	1.23	1675	746	18	0%	1	Yes
ILRT	1.11	1312	574	17	1%	1	Yes
LIGHTT	1.20	1183	473	18	4%	1	Yes
NODCHG	2.10	1541	707	12	0%	4	Yes
NODCOM	1.69	3170	1535	27	4%	8	No (2
NODDED	1.32	1826	865	15	1%	5	Yes
NODDES	1.08	789	346	7	2%	2	Yes
NODDFS	1.17	1492	693	11	1%	4	Yes
NODDPR	1.08	756	295	5	1%	2	Yes
NODDSS	1.00	249	41	3	0%	0	Yes
		410	149				

Fig. 5 Static analysis results

The *standards penalty* is shown as a percentage and indicates the percentage of the code that contravenes the standard. These standards are customised from a standard list that is incorporated into the program. Experience has shown that any modules showing with >5% violation should be reworked.

The *complexity penalty* is measured in a scale of 0 to 10 and indicates parameters such as understandability and maintainability. Basically, the more complex the code, the more likely that bugs will have been introduced.

The *structure* check gives a straight Yes/No result based on matching templates of acceptable structures with the connection matrix of the basic blocks on a module-by-module basis.

DYNAMIC analysis uses regression and development test sequences to check paths and sequencing in a running environment and again has three areas of assessment of test cover – *Code*, *Jumps* (Explicit & Implicit) and *LCSAJ's* (Linear Code Sequence and Jump).

Each is shown as a percentage test cover. *Code* is straightforward executable statements obeyed in sequence, *Jumps* covers control flow branches and *LCSAJ* covers the more complex situations of executable statements sequences followed by control flow branches. A set of results is shown in Fig. 6.

Module Name	Test Files	Code %	Jumps %	LCSAJs %
CPLRT	5	91	89	72
ILRT	3	83	82	69
LIGHTT	1	79	76	50
OVACRT	2	91	83	70
PCST	13	92	87	86
PIT	4	91	85	67

Fig. 6 Dynamic analysis results

Experience has shown that there is considerable variation in the figures across the modules and the breakpoint at which it is worthwhile writing further test sequences varies both with the module and the type of check. Analysis has indicated that realistic targets are 90%, 80% and 70% for code, jumps and LCSAJ respectively, and any divergence from these figures results in an assessment of the test sequences to ascertain whether it is cost effective to extend the testing to give better cover for that particular module.

Overall, the Test Bed phase now pulls out 37% of the post compilation errors and the desk checking 43%.

5 Processor hardware development route

5.1 Background

Range Compatibility. The ICL Series 39 range of machines, announced in 1985, comprised 2 processor designs – internally code named Estriel and DM1. The range was a "compatible evolution" from the previous 2900 series – it was required that all customer software would run without change, but improvements to multiprocessors were made by incorporating nodal architecture where nodes are interconnected via fibre optic cables. Also, I/O channel capacity was improved by the use of similar connections. The SX machine is a further compatible evolution – all customer software must still run without change but further improvements, e.g. higher bandwidth fibre optic technology, have been made to multi-node and I/O connections.

It is worth noting that compatibility can both help and hinder the designer, e.g. the freedom to design a new internal hardware architecture in order to extract the optimum from a new technology is considerably constrained – compared with the situation of SPARC RISC designers who had no such problem. However, one might speculate that difficulties could arise after a few generations of SPARC technological evolution have passed and eroded their original optimisations. The advantage of the SX case is that several

generations of test software exist, with which to validate the new design and, indeed, *ensure* that it is compatible with previous machines.

Technology.
SX logic chips are faster (1.6×) and have higher integration density (3000 gates vs 400 gates), than those in Estriel. The effect of improved speed is self-evident, but other attributes also change:-

First, more gates/chip implies fewer chips/machine, hence reliability is improved. This improvement is compounded by silicon processing improvements which means that a single chip is now more reliable. In addition, the design incorporates careful management of soft and hard failures in RAM devices to provide resilience. The result is that a single node will only see a catastrophic fail once in 2 years. Multi node configurations ensure that even this low rate of intrinsic failure is not allowed to crash a customer system.

Second, the risk of chip reworks after prototype switch-on is increased. With only 400 gates/chip, the Estriel designers were able to minimise the degree of "intelligence" built into the silicon itself. Many of the chip designs were "super MSI" style with much design complexity built into the PCB wiring and microcode. This meant that most of the design errors found at prototype stage could be fixed without changing the silicon. At 3000 gates/chip, the SX style is very different. Chip partitions contain so much more logic (with only twice the pinout) that complexity has to migrate from the PCB level. Hardware design errors now *normally* result in chip rework, hence the design development route placed heavy emphasis on the quality of the design released to build.

Performance
The marketing requirement for SX was to provide an aggressive performance improvement over Estriel. This was to be achieved with a processor technology 1.6 times faster and a store technology 1.4 times faster. In order to bridge the "gap" between the technologies and the performance requirement, the SX design needed to be much more complex, and to encompass much more functionality [Ref. 3]. The use of Picode retained some "soft" control, however many functions previously executed by microcode had to be taken on by the hardware. This added significantly to the difficulty of validating the hardware prior to design release.

A final perspective is that SX was the most powerful general purpose uniprocessor then announced in the world. A large, complex, "hard", design was necessary to implement it.

5.2 *Partitioning [see Fig. 7]*

It is axiomatic that any large design task is partitioned in organisational terms as closely as possible along functional and physical implementational lines – it is a matter of managerial judgement to decide upon the best route to steer through the conflicts arising from this bland statement!

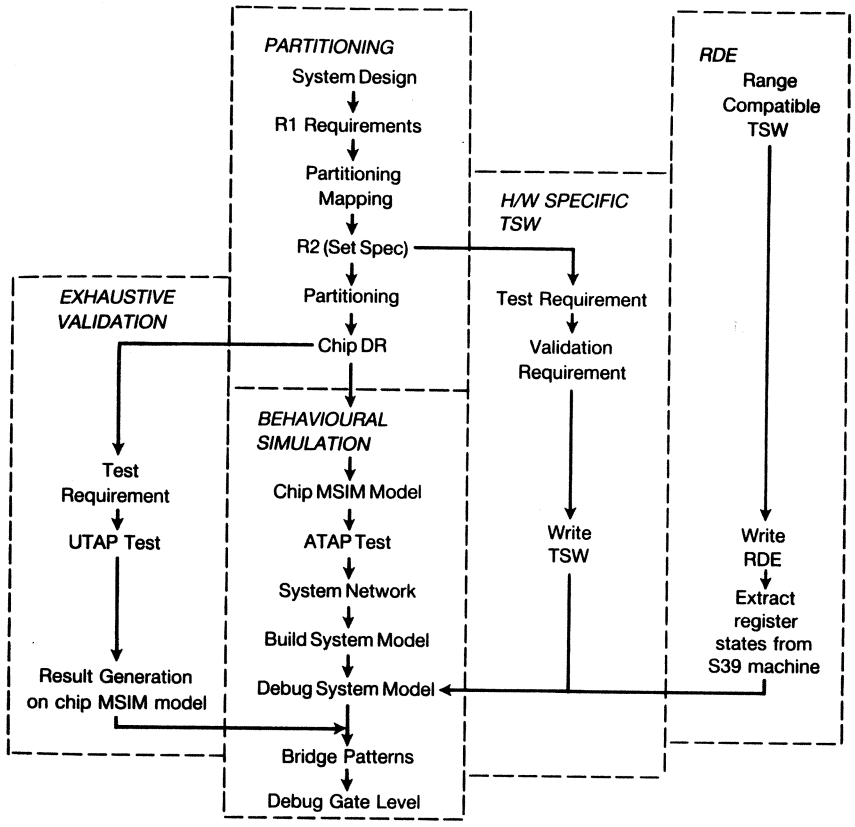


Fig. 7 SX hardware design validation

The SX hardware group was divided into implementation teams of five to six engineers each with a leader. Each team produced, and worked to, a "Set Specification" which encompassed the whole of the functional design unit (Set) for which they were responsible.

As described above [Section 2], the outputs of the system design process were documented statements of requirements – termed "R1". The implementation teams participated fully in the system design and they produced the Set Specification for their own design unit from the relevant R1 documents. This level of partitioning was finished when all R1 requirements had been mapped into implementation (Set) documents. The mapping was audited for completeness.

The Set Specification is the top level document for the team – it details the High Level Design (HLD) of the Set, i.e. the implementation of the R1 requirements and defines the chip partitioning. From it follow:-

Behavioural simulation.
Chip design requirement (CDR) – for each chip.
Test requirements – to allow software writers to test the Set.
Validation requirements – to allow simulation engineers and auditors to complete the validation of each individual chip prior to design release.

5.3 *Behavioural simulation [see Fig. 7]*

A two level approach to simulation was adopted for SX – behavioural and gate. The features of this approach, as opposed to gate level only, include the following:-

Advantages –
a) Behavioural code represents a verifiable specification of the design.
b) Early availability of a working model, enabling design faults to be fixed at lower cost.
c) The designer is able to visualise his design in functional rather than just physical terms, so providing an in-built "desk check".
d) Reduced processing power requirement enables a larger test set to be run.
e) Bridging of test patterns captured around a unit of design executed at behavioural level, into gate level, enables testing of gate level in isolation from the remainder of the design.

Disadvantages –
a) Designers require more tool skills.
b) Extra manpower required for coding.
c) Cumbersome route for design fixes later on.
d) Clock and diagnostic services only exercised at gate level.

The SX designers coded a model of each chip, using an in-house simulator – MSIM. The code was subjected to a style audit, mainly for understandability. Each model was then alpha tested with specific hand written test patterns (ATAP) – this process removed 85% of code faults and a few design faults. The individual chip models were networked together to build the complete system model – this came to 100K lines of code. The subsequent debugging identified 400 more code faults and 550 design faults.

5.4 *Range definition exerciser – RDE [see Fig. 7]*

Although simulation is now used in the verification of every significantly complex ASIC design, it is important to recognise the limitations of the technique. The most obvious of these is performance – one second of real time running of the SX processor is equivalent to one continuous month of behavioural simulation running using all the service machine processing power available to the design project!!

Given that fault situations in I/O, inter node, slave and pipleline, can sometimes take hours of real time to become manifest, it follows that the most test cover possible must be compacted into the least time. In practice, the limit on the size of the complete simulation test suite is a few million clock cycles.

Given the complex and "hard" nature of the SX design, the engineers wanted to run tests equivalent in cover to an existing range compatible test suite – known as Range Definition Tests (RDT). Unfortunately, RDT was, and is, well beyond our simulation capability – a billion clock cycles would take 10 years to run, without iteration for any fixes!!

However, it was estimated that the ratio of target test instructions to data generation and checking instructions was approximately 1 to 10 000. This suggested the possibility of removing most of this overhead by running only test instructions on the simulation, whilst generating data and checking results in real time on a Series 39 host machine.

The resulting tests became known as Range Definition Exercisers (RDEs). The method of checking the results relied on the ability of the Level 80 microcode to monitor register states at the completion of PLI instructions.

The RDEs were run on a dedicated Level 80 with the monitoring microcode enhanced to write the PLI visible registers, at the termination of each PLI instruction, to a dedicated area of the main store. This area of main store was then dumped to magnetic tape and transferred to the service machine for processing. Subsequently, this data could be used, together with the System simulation model, to compare Level 80 and SX visible register sets at the end of each PLI.

There are 17 RDEs, 8 testing PLI and 9 data dependent tests. The PLI tests took a total of 150K clock cycles whilst the data dependent tests took 600K cycles for single and double length arithmetic, or 7600K cycles if quad length was included as well.

5.5 *Hardware specific test software [see Fig. 7]*

Each Design team produced a Test Software Requirements document identifying areas of their part of the SX design which were not tested by the RDEs or other tests. The Test Software Group responded to this document with a specification of the tests, which was then reviewed with the project before low level program specifications for the Test Software were produced. As the tests were specific to SX they could not be validated fully prior to use on the System simulation.

5.6 *Exhaustive chip validation [see Fig. 7]*

A feature of running PLI level test software is that a homogeneous test cover of every hardware function is unachievable – for example, some gates in the

arithmetic unit may be exercised in nearly every clock cycle in the test, whilst some gates in an error management function or slave control may only be exercised once after millions of clock cycles of testing. Any efficient test strategy must address these "sparse" areas. The hardware specific test software (section 5.5) allows for testing the design at a lower level of detail than PLI – using Picode. However, this still leaves some areas of the design with little cover.

These areas in SX were identified from the chip design requirement in a test requirement document which is chip specific. Bit patterns were applied to single chip MSIM models and gate level models, and the results compared. These tests were known as Unit TAP (UTAP).

5.7 The future

The foregoing was intended to give an insight into the complexity of a modern processor, and into the innovation required to design and validate it. Future designs will have even tougher targets to meet, especially reduced time to market. The ultimate challenge is to switch on a prototype free of any hardware deficiencies – but a more modest step would be to switch on a million gate design with only 10 faults!!

6 Configuration management

At the start of the SX Project, the need was envisaged for a system to allow a high degree of control in the manipulation of large volumes of design data through many overlapping development steps. Such a system had to provide facilities to allow effective control at both design team level, and at system integration level. In addition, the system needed to be capable of integrating a large number of different design tools into a uniform, interactive, and easily usable environment.

At that time no suitable commercial tool was available to meet the project requirements. The SX Design Database (SXDB) was, therefore, developed to support both hardware and PICODE development. SXDB now provides a high level of control, tools integration, data accessibility and performance. In most of these respects, SXDB is still ahead of commercially available systems.

The basic facilities of this system are listed below:

1) Product/sub-product version control
2) Audit trail facilities
3) Integrated documentation system
4) Tools invocation
5) Edit facilities
6) Interactive report/enquiry facilities

Each of these facilities is described below.

6.1 Product/sub-product version control

The versioning mechanism used within SXDB was initially invented for the CADES database, used in the VME development route [Ref. 4]. (The ICL DDS system has also borrowed some of these concepts). The basic philosophy is one of having a version state for an entire sub-product. The version state of any sub-product is an amalgamation of objects changed at that version, plus all other objects "inherited" from lower versions.

The creation of new version states for a sub-product are determined at team level as needed by the phased development strategy. Mechanisms are provided to lock and freeze version states, to stop further change at any particular version.

A "System Version" mechanism allows a single version of each sub-product to be related to form a single version of the total SX product. The combination of System Version and sub-product version facilities allows flexible control at both System and team leader level; in particular, System simulation models can be built and maintained for any development stage, unaffected by the later sub-product development.

6.2 Audit trail facilities

Each version of an object stored/modified in the SX database has a set of control details associated with that object. Such details include: date/time created, who created it, and intermediate freeze state. In addition, most objects are given a textual description for every change applied.

6.3 Integrated documentation system

All documents within the SXDB are assigned their own versions. Design documents are directly related to objects within the design hierarchy, and mechanisms are provided to record the impact of changes arising from either documents or objects.

6.4 Tools invocation

There are two basic classes of tools controlled via SXDB: "loosely"-coupled tools, such as the network editors running on graphics workstations, and "tightly"-coupled tools. The "tightly"-coupled tools are all invoked directly via the SXDB menu interface common to the majority of other facilities, including checks, reports, etc.

6.5 Edit facilities

Facilities for editing of file based design data are provided by SXDB. These facilities allow the current design state to be checked-out into a scratchpad filestore, edited and checked-in to the database.

For graphical data the file is transferred to a graphics workstation for edit. For textual data, VME edit facilities are utilised under direct control of SXDB.

6.6 Interactive report/enquiry facilities

A comprehensive set of interactive reports and enquiries is included within SXDB. Each of these is available via the SXDB menu interface, and provides fast, online response. Enquiries provide details for a single database object; reports provide similar details for multiple database objects selected via common attributes.

7 Measurement within the development process

There were two basic reasons why it was decided to apply extensive measurement to the development processes. First and foremost, it allowed assessment of whether the development was on schedule and whether the assumptions made about the many design parameters were still valid.

Second, measurement allows identification of the key factors that control the process and ensures they are properly applied at the planning stage of the next project. Many of the initiatives this time were based on measures taken on the previous development.

The prime measures throughout the development process were directly related to the delivered quality of the product. However, another key measure is the amount of effort expended in each phase of the development route. This directly affects the time to market and needs to be understood in detail if management is to make the right decisions on the best use of resources.

Taking a simple breakdown of the development route for NSX firmware, the synthetics used in the initial planning were as shown in the table below.

Devt Phase	HLD	LLD	CODE	DESK CHECK	TEST BED	M/C TEST
% Total effort	24	24	10	2	25	15

A user friendly database application was developed that would allow individuals to enter details of how they spend their time, and provided direct correlation with the planning parameters.

The WORK STATS database provided over 90 separate categories structured in a three level hierarchy. Input was extremely simple and checking facilities ensured all entries were valid.

The database has been in use since October 1987 and covers the full development cycle of several projects.

Extensive analysis facilities were built in allowing extraction of the data to be output for individuals or teams, for any category, and over any timeframe. Later, incorporation of spreadsheet facilities allowed output to be displayed in various graphical forms such as that shown in Figure 8.

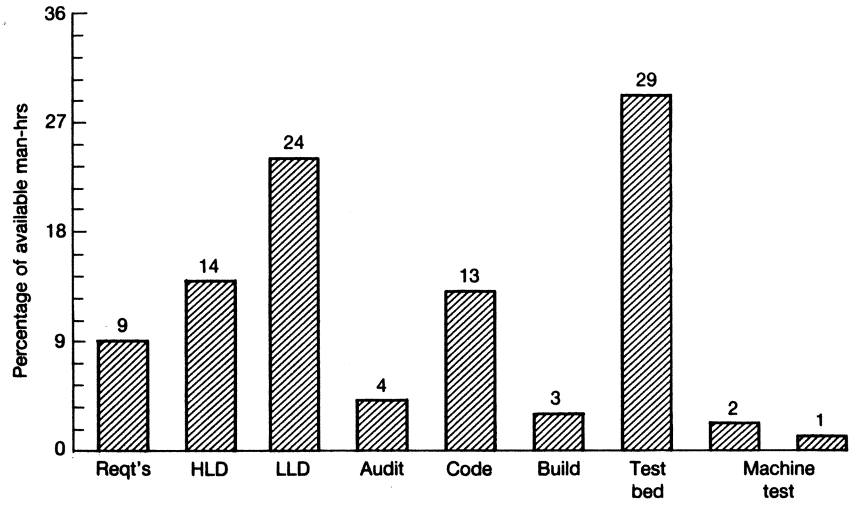


Fig. 8 Effort profile — application firmware development

This gave the breakdown of effort against each of the development phases, allowing the planning synthetics to be monitored directly. Comparison with the initial planning synthetics shown above highlighted variations that needed to be assessed.

Particular points to note are that the effort prior to coding accurately matched the synthetic and, more dramatically, the effort required for on-line machine validation was reduced from the expected 15% to less than 4%.

8 Concluding remarks

The development routes cover an amalgamation of tools, methods, checkpoints, metrication and design control. The successful implementation of this methodology however requires discipline and commitment from each and every member of the team. This commitment has resulted, to date, in an order of magnitude improvement in the delivered quality of the products.

The processes described are a few of the many involved in the development of SX. The challenge with each mainframe development is to understand our

processes better and identify where changes to the methodology, techniques or tools can lead to the levels of improvement needed to maintain our position at the forefront of mainframe development.

The examples given clearly show that, by careful analysis of the previous development cycle and by assessment of the advances made in the software packages available, considerable improvement can be made in the next development cycle. It is this challenge that ensures designers, in whatever discipline, will have an interesting and exciting future.

Acknowledgements

The work described reflects the efforts of all members of the SX project team.

The authors would like to thank everyone involved for their efforts and their help in preparing this paper.

References

1 Birtwistle G.M. Discrete Event Modelling on Simula. Macmillan Publishers Ltd. ISBN 0–333–23881–8.

2 Ward P.T. and Mellor S.J. Structured Development for real time systems. Prentice Hall ISBN 0–13–854787–4.

3 Allt G, Eaton J.R. and Hughes K. The SX Node Architecture. ICL Technical Journal Vol. 7 Nov 1990. 197–211.

4 Snowdon R.A. "CADES and Software System Development". Software Engineering Environments, North-Holland Publishing Company, 1981.

Physical Design Concepts of the SX Mainframe

C. Shaw

Corporate Servers Product Group – ICL West Gorton, Manchester, UK

Abstract

The spectacular improvements in silicon chip circuit integration densities and speed, which have now been sustained for over two decades, are well known. The effective exploitation of modern LSI technology in a high performance mainframe computer places challenging demands on the hardware and packaging which supports the electronic subsystems.

This paper describes the implications of these demands on the physical design of the SX mainframe and outlines some of the resultant design solutions. Contents of the major functional hardware sub-systems within the SX node are summarised.

1 Physical design requirements

In order that the machine can be readily adopted by existing customers whose installations require increased computing capacity it is important that the new machine places no greater demand on its environment than its predecessor (typically Series 39[1] Level 80 systems in this case). The parameters of particular importance are:-

Overall Dimensions

Compactness brings numerous benefits: for the customer a reduction in the computer's footprint provides opportunities for more convenient siting of equipment as well as efficient use of premises. For the manufacturer delivery is made easier and transportation costs are minimised. In the case of the SX mainframe by restricting the overall height to 1500 mm it becomes possible to use standard air-freight containers for shipments to customers outside the UK.

Mainframe installation can also be simplified as the overall dimensions reduce. If the complete mainframe can be transported in one piece the need for on-site re-assembly is eliminated. Moving the equipment within the customer site, when purpose-built premises are not available; eg up lift-

shafts, through narrow corridors and door openings etc., is made more straightforward.

The overall dimensions of the SX mainframe are 1760 mm length, 770 mm width, and 1450 mm height.

Operating temperature range
Whilst component reliability benefits from maintaining low transistor junction temperatures, it is desirable to permit as wide a range of ambient operating temperature as possible. Imposing narrow limits on the operating range requires excessively tight controls on the computer room air-conditioning and results in increased running costs. The permitted range should ensure that at low room temperatures there is no risk of condensation forming within the mainframe when the humidity levels are high, and at the upper limit the temperatures of high dissipation components remain within their design targets. SX nodes are able to operate through an ambient temperature range of 15–29°C.

Acoustic noise emission
Our design objectives are to produce mainframes which have noise levels comparable to equipment, such as Workstations, designed for use in office environments (typically 55–60dBA sound pressure when measured at 1000 mm). This helps to ensure that overall computer room ambient noise, when many units such as disc sub-systems and printers are located together, is kept within comfortable limits.

Electrical power consumption
For a chosen computing performance the electrical power consumption is primarily dictated by the choice of circuit technology. Even so, careful design of the internal power supply and cooling systems for maximum efficiency can contribute significant energy savings which directly benefit the mainframe's lifetime operating costs.

2 Performance Implications

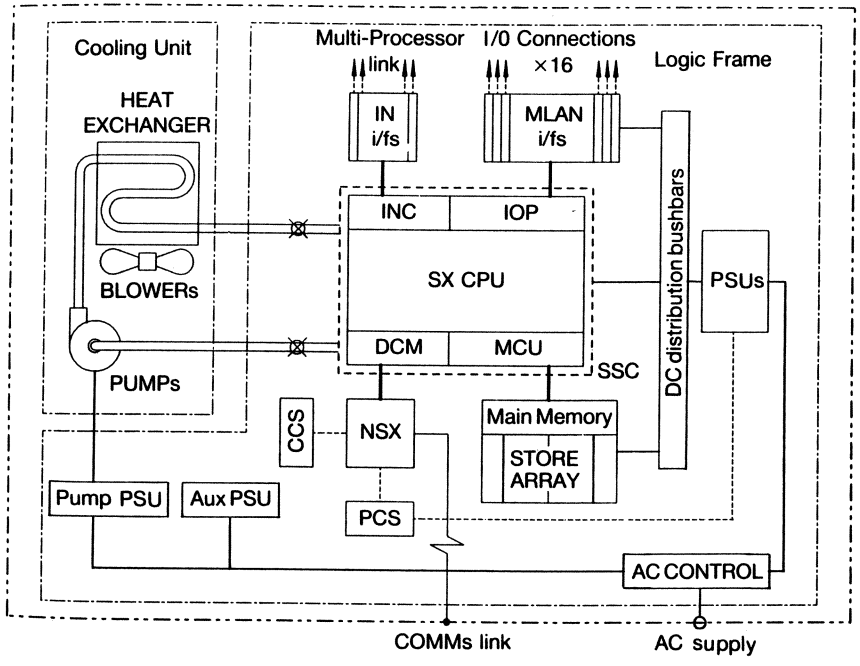
The reduction in logic cycle times needed to achieve the performance of a modern mainframe has required not only the improvements in silicon chip processing but also comparable changes in the packaging of the logic circuit assemblies to reduce interconnection path lengths. The speed of light remains a constant impediment in the race for ever more MIPs!

Component densities on the logic assemblies have increased as a result of reductions in package size and finer trace geometries together with more layers on the printed circuits. The logic assemblies are more closely spaced to shorten signal paths. All these factors tend to result in increased heat density and dissipation within the logic units. In order to maximise the intrinsic component reliability it is imperative that these higher dissipations are managed without increasing the component operating temperatures.

For maximum mainframe peformance Emitter Coupled Logic (ECL) remains the silicon technology of choice. This is a power hungry logic family. The circuits used for the SX central processor unit (CPU) have a speed-power product of 0.4 picoJoule/gate^2 which when packaged at the gate densities achieved on the SSC exceed the heat transfer capacity of a forced-air cooling system.

3 Technology overview

The major functional components and interfaces of the SX Node are shown in Fig. 1 and a summary of each unit's relevant characteristics follows.



SX NODE - Functional layout

Fig. 1 SX Node Functional layout

3.1 CPU

In the SX design it has proved feasible to extend the "Single Board CPU"[3] concept adopted by Fujitsu and incorporate not only the CPU but also the performance-critical I/O and Memory Control sub-system logic on the same pcb. This has major advantages in reducing transfer time delays on the associated interfaces.

The Sub-System Carrier (SSC) forms the heart of the SX CPU. The SSC technology is a variant of one developed by Fujitsu Limited for use in

their M780 range of mainframe computers. The logic and interconnections are entirely specific to the SX computer and wholly designed by ICL.

The SSC comprises a 540 mm × 488 mm 42 layer printed circuit assembly on which are mounted up to 336 ECL LSIs. This assembly is sandwiched between a pair of Conductive Cooling Modules (CCM)[4] which transfer the heat dissipated by the logic circuits. Each LSI has a nominal power dissipation of up to 9 watts and hence the CCM pair has a heat transfer capacity exceeding 3 Kwatt.

A maximum of 700 logic signals are provided to connect the SSC with other functional units in the SX Node. These signals are carried by miniature coaxial wiring to ensure low electrical noise and arranged into 14 multiway ribbon cable and connector assemblies.

3.2 Main Store Unit (MSU)

The MSU is designed to use Dynamic Random Access Memory (DRAM) components having typical access times an order of magnitude greater than the CPU clock cycle. From an SX logic designer's viewpoint the DRAM component timing tolerances (skew) arising from normal semiconductor manufacturing variations exceed the logic beat. Within the Main Store system, in order to minimise this skew, which represents wasted potential machine performance, it is desirable to both shorten and equalise the signal paths to and from the DRAM elements.

In the case of the SX main store skew has been optimised by the design of a double-sided motherboard assembly as shown in Fig. 2. All store dataflow

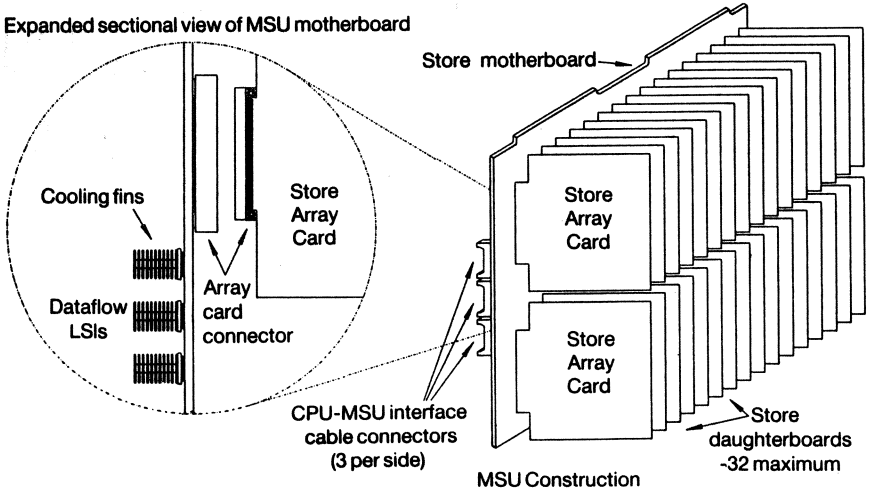


Fig. 2 MSU construction

circuits are arranged on one face of the motherboard and connect via short signal paths to the Array card connectors on the opposite face.

An "active" motherboard containing ECL dataflow multiplexing and direct drive to all Store Array cards was designed to maximise performance.

High storage capacity within the available volume is achieved by using double-sided surface-mount technology for the Array cards which contain the DRAM components. The fully configured SX Main Store contains 32 array cards.

By limiting the maximum utilisation of the logic circuits within each LSI on the MSU, and arranging these devices in only three rows across the full width of the motherboard, the power dissipation is restricted and it is possible to use an air-cooled variant of the CPU LSI technology.

Airflow through the MSU required careful design. The dataflow LSIs on the motherboard and some regions of the Array cards need high cooling capacities. Reduced air speeds arising from the additional volume created by the absent Array cards when the MSU is depopulated as in minimum configurations, must not degrade the motherboard cooling.

3.3 External interfaces

With the exception of the AC mains supply cable and the communications line linking to a conventional modem for remote maintenance (Telesupport), all interfaces between the Node and other SX system components (eg discs, printers, workstations) are provided using high speed serial fibre-optic connections. This greatly simplifies potential cable shielding and earth-loop problems and reduces the bulkiness of the interface cables and connectors.

Optical fibres (Macrolan)[5]
The proprietary ICL optical fibre connection system introduced when Series-39 was launched is also retained for SX systems. It enables processing nodes and the I/O controllers within a system to operate with separations of up to 2 Km, thereby facilitating remote operation and dispersed systems for "disaster proofing" etc..

3.4 I/O & Inter-Node Couplers

Macrolan is used both for I/O Controller ↔ Processor connections and inter processor links in multi-node systems.

Inter processor data is transferred by a 200 Mbps variant of the Macrolan normally used for I/O traffic. A special purpose optical fibre link is also used to maintain data synchronism among each of the Nodes in a multiprocessor system. This "Transmission Sequence Number" (TSN) system employs the same optical interface components as does Macrolan.

3.5 Node Support Computer (NSX)

The NSX takes care of Initialisation, Self-test, Reconfiguration and Remote (diagnostic) access for maintenance. All development commissioning takes place using this mechanism – many of those customers seeing SX during visits to West Gorton have been surprised by the absence of engineers around the prototype machines, it being more efficient and comfortable to access the systems from terminals in the design offices.

The NSX and the Power and Cooling Control Systems (qv) are separate microcomputer units which operate independently from the main CPU. Power to these units is supplied by dedicated Auxiliary PSUs so that control functions are available whenever AC is applied at the node.

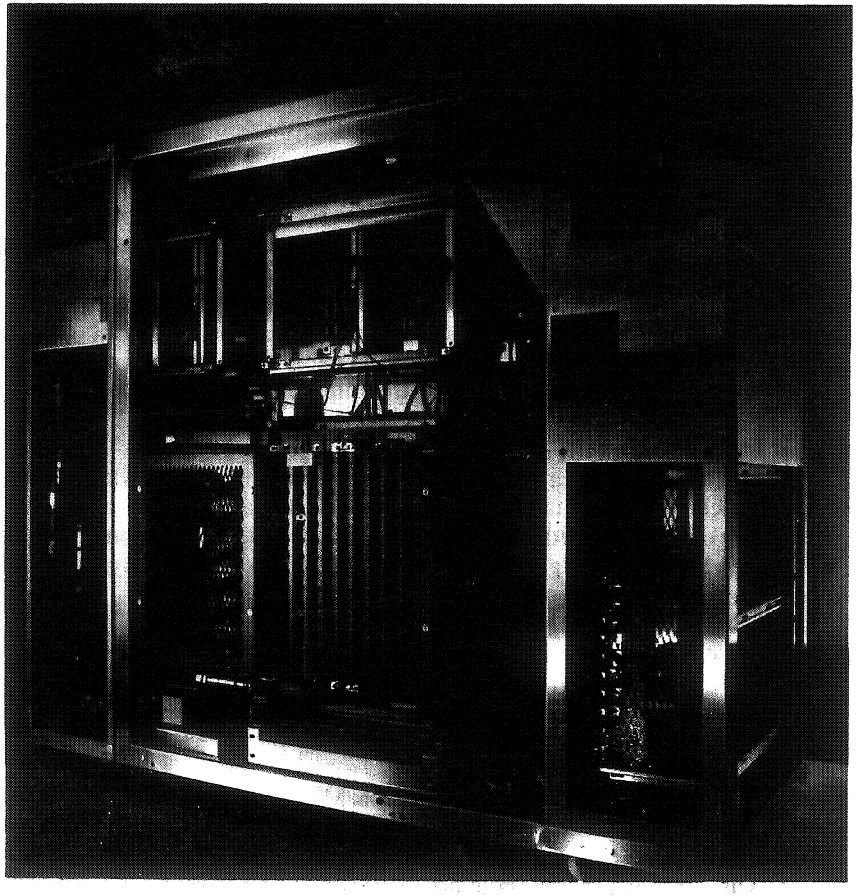


Photo 1 SX Node – internal view

4 Reliability

The advances in silicon technology have produced dramatic improvements in the reliability of computers. Electronic component failure rates are now typically better than 1 in 10^7 hours. Customers now expect their mainframes to function continuously for months without service breaks. It has become essential to ensure that the electro-mechanical sub-systems used for powering and cooling the CPU achieve comparable reliability.

4.1 Resilience

The basic strategies for ensuring high reliability of the node physical design sub-systems have been to eliminate electro-mechanical components as much as possible and, where this could not be achieved, to provide functional duplication. Examples of these strategies are detailed in subsequent sections.

5 Accessibility

Previous members of the Series 39 Processor family achieved very compact performance/volume ratios at the expense of ease-of-access to the individual logic assemblies. The high reliability of these designs meant that this was not a customer perceived problem, although for ICL increased system upgrade and enhancement times resulted.

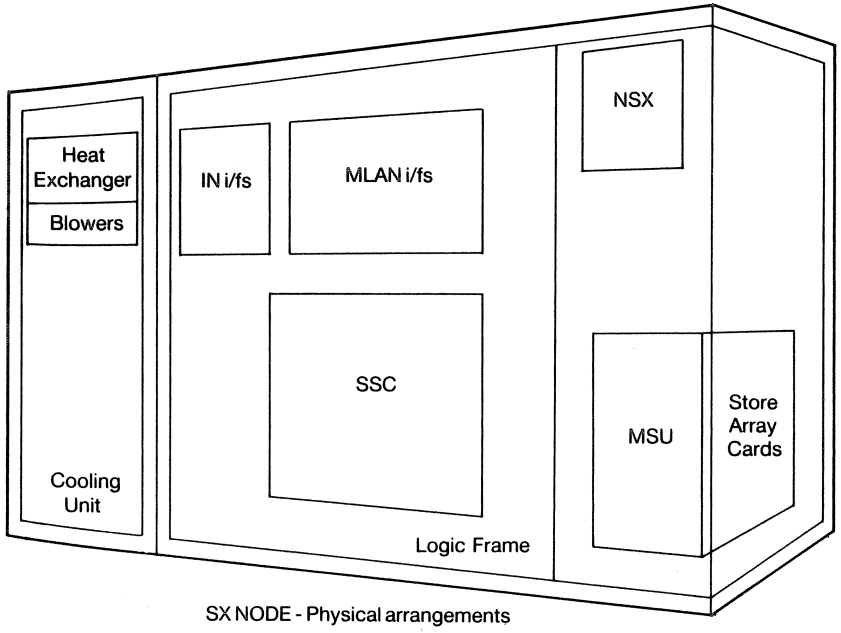


Fig. 3 SX Node – internal arrangement

With the SX node packaging design much care was applied to ensure that each field-enhanceable logic assembly could be accessed for upgrade/replacement without needing to remove other sub-systems. This accessibility is evident from Photo 1.

6 Power distribution

The major logic sub-systems in the SX mainframe are supplied from four low voltage DC rails. Rail currents of up to 750 amps are catered for. Voltage regulation has to be carefully controlled to eliminate noise pick-up on the logic interconnections.

As well as the large component timing variations which occur, process spreads in semiconductor manufacture result in substantial variations in circuit power consumption – ±30% of nominal is not uncommon. This poses a problem for the computer packaging designer who has to provision power to meet the worst case demand, whilst also seeking to minimise the overall machine size.

In order to minimise component dissipations and maximise logic gate switching speeds, semiconductor designers are progressively reducing both the logic switching levels and the operating voltages for the circuits. This trend further complicates the packaging design as high current PSU conversion efficiencies tend to diminish at low output voltages and noise susceptibility increases as logic switching levels reduce. Great care is needed to ensure that the DC distribution design minimises any voltage differences across the logic assemblies in order to preserve the noise immunity of the digital circuits.

6.1 Power Supply Units

All main DC rails in the SX node use the same switched-mode Power Supply design. Each unit can supply over 800 watts of regulated low voltage DC. Up to 15 PSUs are installed in a fully configured Node.

High AC to DC conversion efficiencies are achieved by the use of power Field Effect Transistors (FET) in the inverters. Switching frequencies are ×5 higher than is feasible with conventional junction transistors. The very short FET on/off transition times greatly reduce the switching losses and the increased switching frequency allows the use of smaller magnetic cores in the wound components. In combination these factors allow the PSU volume to be reduced whilst maintaining adequate cooling airflows. The output power density achieved overall by the PSU approaches 0.2 Kwatts/litre.

PSU efficiency is an important physical design parameter as the conversion losses can account for a significant proportion of the total cabinet dissipation. As the DC rail voltages reduce, voltage drops across the PSU output rectifiers form an increasing proportion of the high current circuit and hence the efficiency decreases.

One complication arising from the higher switching frequency is the potential for increased Radio Frequency Interference (RFI). The high inverter voltages and very fast switch rise and fall times can result in a broad spectrum of electro-magnetic-radiation from the PSUs which need careful design and shielding to avoid RFI propagation.

All PSU connections, ie AC mains input, DC output and the control interface are pluggable. This reduces system assembly and replacement times and eliminates the need for tools in these processes. By placing all electrical connections on the inner face of the PSUs the usual protective cover plates are no longer required and effective RFI shielding becomes simpler.

The Power Supply Unit nominal DC output voltage can take one of four values (between 2 and 5.5 volts) according to the setting of link connections on its control interface. This allows any unit to be used on any voltage rail without the need for set-up adjustments, thus simplifying spare parts inventories and eliminating the risk of mis-connection. The links are hard wired at each interface connector in the PSU mounting rack to provide a "position selectable" voltage setting.

Three phase line-to-line AC is supplied to every PSU in order to ensure that the line currents in each phase are balanced and for increased resilience to transient supply disturbances affecting one or more phases.

6.2 DC busbars

The DC distribution from main PSUs to SSC and MSU takes the form of a multilayer planar construction, rather like a heavy-duty pcb, which runs along the centre of the logic frame. Fig. 4 illustrates this with a sectional view through a PSU output connector. The conductor for each voltage rail is interspersed with an insulating laminate and the complete sandwich forms a very stable matrix in which the low impedance DC connector sockets can be accurately located. The configuration of insulating washers and conductive collets shown in figure 4 ensures that the connector socket is securely clamped to make good contact with the selected busbar layer whilst remaining well insulated from all other DC rails. Our measurements have shown that this method, in conjunction with suitable connectors, results in no greater DC voltage drops than would arise using conventional braided flexible links bolted to PSU terminals.

Every DC rail contains one more PSU than is required to supply the worst-case current demand. This "N+1 Resilience" ensures that node operation is not disrupted should a PSU fail in service and so replacement of any faulty units can be deferred until a convenient maintenance opportunity occurs.

6.3 Power Control

The performance of the PSUs and the DC rail conditions are continuously monitored by a dedicated microcomputer sub-system. This unit also pro-

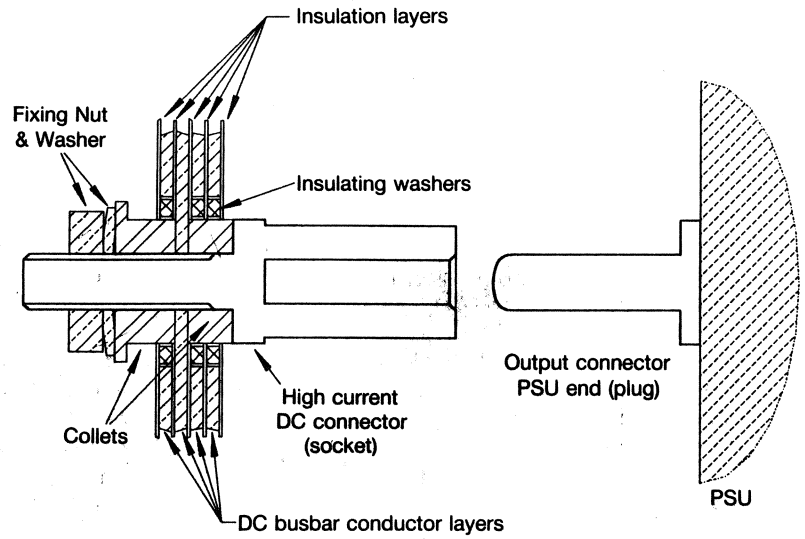


Fig. 4 DC busbar construction

vides continuous control of "current sharing" – a technique which ensures that all active PSUs on any voltage rail are delivering the same proportion of the total current demand. When PSUs are connected in parallel such a technique is necessary to compensate for minor differences in the load impedance seen by the individual units which otherwise would result in imbalances driving some PSUs into current-limit and others into no-load conditions.

The power control system (PCS) also controls voltage sequencing during node switch-on and switch-off and has a communications link to the Node Support Computer. On successful completion of the power-on sequence the PCS uses this link to initiate logic startup of the SX node. Power system fault diagnostics are reported to the remote maintenance system via the NSX link.

In the event of a supply or power fault condition causing a node shut-down or preventing startup, the power system status and any diagnostic signatures are stored in a battery powered memory until normal reporting can be resumed.

7 Cooling

7.1 Cooling technology selection

The heat transfer capacity of any coolant is proportional to mass flow per unit time in contact with the items being cooled. For maximum heat transfer

within a particular coolant volume liquids are greatly superior to gaseous coolants. However the simplicity of forced air cooling (eg no containment or insulation problems etc.) leads to its general use in preference to liquids but as the component dissipations rise the air speeds and component fin dimensions have to increase to unacceptable levels. Acoustic noise also increases with higher air speeds and design of suitable blowers becomes impractical.

7.2 Cooling Unit functions

Design considerations
Although the CPU technology dictated that liquid cooling was required, a key objective was to ensure that this did not make installation, maintenance or operation of the SX mainframe more difficult for our customers. Our aim was to achieve a design in which the customer's premises provide electrical power and normal computer room air-conditioning without any other awareness of the cooling technology used within the mainframe.

Most important was the desire to eliminate dependence on a direct connection to the building air-conditioning plant. Such a linkage, which is sometimes used in other designs of liquid-cooled mainframes, would constrain the installation of the node in buildings with minimal heating & ventilation controls and could drastically impact the computing system reliability because of plant failures/outages unrelated to the node.

Simplicity of operation
The cooling system should not impose a greater level of user skills or intervention than is required for the correct functioning or control of the mainframe's logic circuits. Unattended operation and deferrable maintenance of any defective components are therefore essential.

Reliability – Duplication of system-critical cooling components
Coolant pumps and their associated power supplies are duplicated to avoid system outages in the event of malfunction. Only one pump is in use at any time, but regular changeover of the pump in operation occurs under normal conditions. This ensures that latent faults are not hidden.

For each functional grouping of blowers in those areas of the node requiring forced-air cooling, "N + 1" resilience is provided so that adequate cooling air flow can be maintained should a blower fault occur.

For both the liquid and air circuits the design must prevent back-circulation of the coolant under pump or blower fault conditions.

Water quality
To achieve a reliable operating life (which could be as long as a decade) it is important to eliminate sources of contamination of the coolant. Prior to final assembly all the piping components undergo a rigorous cleansing process

which includes an extended period of rinsing in an ultrasonic cleaning bath. The water used in the coolant system is de-ionised and purified to pharmacological standards.

These measures minimise the inclusion of unwanted particulate debris and biological contaminants. In consequence there is no food supply and, after initial filling and system switch-on, any anaerobic bacteria are quickly starved and die. The growth of aerobic bacteria is inhibited because the closed-loop coolant recirculation does not provide an interface for re-oxygenation of the water.

Materials selection
All materials in the coolant circuit must be carefully selected to avoid corrosion. This consideration includes not only the pipes and pipe-fittings but also such items as the constituent metals used in any brazed joints so that electrochemical reactions are minimised. It is also important to control the composition of plastic materials such as flexible hoses, valve and pipe seals etc., as many of these items normally contain compounds from which undesirable ions such as halides can be leached into the coolant. Materials used in the piping manufacturing processes such as fluxes and acidic cleansers also require evaluation.

Initial SX Cooling Unit prototypes were manufactured using a stainless steel piping system, but considerations of materials availability, compatibility and manufacturing processes led to the subsequent choice of copper based systems.

Extensive testing
Validation of the cooling system design has involved extensive testing, including trials of materials compatibility, component performance, water quality, and mechanical integrity as elements of a program designed to determine the lifetime behaviour. Where possible, test conditions in excess of normal operating conditions have been used to accelerate the detection of long-term effects.

Periodic maintenance
To ensure that optimum coolant quality persists throughout the life of the node a simple annual maintenance procedure has been devised. The cooling circuit contains a pluggable cartridge filter which can be replaced without interruption to the normal operation of the node. Whilst the filter cartridge is being renewed an ion-exchange unit is installed in place of the filter. This purges any ionic built-up and restores the coolant to its initial condition.

Control – Pressure & Flow
Incorporation of all the liquid-cooled sub-systems into a single assembly has simplified the coolant circulation design by ensuring that only a fixed flow rate would need to be provided. During initial design the option of providing a free-standing cooling unit capable of supporting two or more nodes in a

multi-processor system was studied. Such an option would have reduced the cost of liquid cooling per node but was rejected because of the design and logistics complications that resulted.

The coolant circuit pressure and flow rates are factory preset during final assembly and test. Each pump is powered by its own AC-AC converter. This isolates the pump performance from the effects of changes in the mains supply voltage and/or frequency.

7.3 Cooling Control

In a similar method to the Power Control System an autonomous microprocessor based unit is used for control and monitoring of the cooling system functions. The Cooling Control System (CCS) continuously monitors coolant pressure, flow, and temperature. Abnormal readings can invoke blower speed changes to increase cooling airflows or pump changeovers as required. If the abnormal conditions persist and exceed preprogrammed limit values the CCS will shutdown the node to prevent possible damage.

Coolant conductivity is also monitored to check that corrosion or chemical contamination are not occurring.

Rotation speed sensing circuits are fitted on all blowers and monitored by the CCS. This provides for accurate resolution to the faulty unit when blowers are operating in parallel and ensures detection and fail reporting occurs before there is any risk of overtemperature conditions.

System protection
Whenever AC mains is reapplied to the mainframe the CCS performs a series of integrity tests to confirm that the all system cooling functions are operational. Only when these checks are completed and the node environment is satisfactory (eg the inlet air temperature from the computer room is within specification) does the CCS enable an interlock to allow DC power to be applied to the logic circuits. This interlock has "fail-safe" features so that any subsequent condition leading to loss of control will cause the DC to be removed.

Cooling Unit sensor data and operating status readings can be interrogated remotely via the NSX. Fault signatures are passed onwards via this route to the normal system diagnostics reporting processes. In this way, the same levels of maintenance information as exist for the CPU hardware and software, are made available for the physical design.

8 Construction

Photo 1 illustrates the internal arrangements of the SX node as seen looking onto the main logic bay. (The opposite side of the unit houses blowers, AC control, and the node power supplies). For clarification figure 3 identifies the main functional units visible in Photo 1.

8.1 *Steel frame*

The logic frame is made from welded stainless steel sheet. This permits ease of fabrication and provides great strength. Additional benefits from the use of stainless steel include the elimination of surface treatments and the provision of continuous electrical contact with the external covers for RFI screening is simplified. By suitable positioning, the frame members also function as air-cooling ducts and mounting brackets for attaching the logic units.

In contrast, a tubular welded frame construction is used for the cooling unit to give all-round accessibility during final assembly of the piping system. This construction was also dictated by the need to maximise the unobstructed cross-section for the heat-exchanger and the airflow through it.

Both frames are bolted together during the final stages of assembly and subsequently treated as an indivisible cabinet.

8.2 *External covers*

In addition to their obvious styling role, the cabinet covers perform important functions including attenuation of acoustic noise and RFI, inlet air filtration and the dispersion of exhaust air.

The panels around all sides of the node are hinged and open for maximum ease of access. Access "under the covers" is only required for engineering purposes and so in normal operation the panels provide isolation from any voltage and current hazards. All cooling air used by the node is drawn into the cabinet through low level grilles around all sides of the unit. This positioning of the inlets avoids the need for under-floor air feeds and hence the SX design can operate with much shallower raised floors than previous designs. The air-grilles are detachable and incorporate foam filter pads which can be exchanged or cleaned by an operator without needing to open hinged side panels in the node.

The transparency needed for drawing cooling air into the cabinet conflicts with the requirement for attenuating RFI efficiently. Prevention of RFI emission is achieved by providing a continuous electrically conducting screen enclosing all the electronics. Any holes in this screen, such as may be required for cable connections or display panels, must be kept to a minimum as they can behave like slot-feed transmitter aerials for HF signals. As logic switching speeds increase a broader spectrum of emission frequencies occurs and to minimise leakage all unavoidable apertures must be kept as small as possible.

8.3 *Use of materials*

In general the expected manufacturing volumes lead to a construction based on use of sheet metal piece parts. The high tooling costs for moulded plastic components are seldom amortized by the resultant reductions in parts costs. However there are areas where the component complexity or finish is best realised by plastic mouldings. One such example in the SX node is the inlet air grille. Fourteen air grille panels are fitted around the node and for this part the quantity involved in conjunction with the intricate surface shapes of the louvres made the development of a low temperature Noryl moulded part worthwhile. The use of a moulded component enabled styling features and retainers for the air-filter pads to be built into the basic part.

9 Conclusions

The technology demands for high dissipation densities and compact signal interconnects in SX have been reconciled with the needs for ease of manufacture, installation and maintenance to produce a mainframe having one of the best performance to volume ratios available. A system having more than four times the processing throughput of its predecessor has been packaged within the same overall volume.

The next challenge will be to maintain the advances in compact packaging for mainframes whilst meeting all the support needs for the next generation of very high performance digital circuit technologies.

10 Acknowledgements

The physical design solutions incorporated into the production model of the SX mainframe are the combination of the ideas from numerous ICL staff throughout Mainframe Systems and Manufacturing & Logistics divisions who have been involved with this project. The final result would have been much less satisfactory without their enthusiasm.

References

1 CAMPBELL-KELLY, M.: ICL Company Research and Development, Part 3: The New Range and other developments. ICL Technical Journal pp. 781–813 (Nov 1989)
2 OHNO, K. et al.: Semiconductor Technologies for FACOM M780. FUJITSU Scientific & Technical Journal, 23, 4 pp. 216–225 (December 1987)
3 NISHIHARA, M. et al.: Single Board CPU Packaging for the FACOM M780. FUJITSU Scientific & Technical Journal, 23, 4 pp. 226–235 (December 1987)
4 YAMAMOTO, H. et al.: Cooling System for the FACOM M780. FUJITSU Scientific & Technical Journal, 23, 4 pp. 243–254 (December 1987)
5 STEVENS, R.W.: Macrolan: A high performance network. ICL Technical Journal pp. 289–296 (May 1983)

Architecture of the DRS6000 (UNICORN) Hardware

G. Poskitt

Advanced Servers Product Centre, ICL Office Systems, Bracknell, Berks.

Abstract

UNICORN is a multi-user, multi-processor UNIX system implemented in RISC (Reduced Instruction Set Computer) technology. Cache memories are provided to enable the very high speed of the processors to be fully exploited, the multiple processors to share, efficiently, a common memory sub-system and the I/O traffic to be buffered. The paper describes the basic architecture of the system, in particular the caching mechanism and the means by which the various physical cache memories are kept consistent among themselves and with main memory.

The UNICORN product is a development by the Advanced Servers Product Centre which is part of the Servers Product Group, which itself is part of ICL's Office Systems Division.

UNICORN is a multi-user UNIX system designed to meet the requirements for office applications for the nineties. The processing element of UNICORN is based on SPARC (Scaleable Processor ARChitecture) RISC technology.

One of the key features of UNICORN is scaleability. It has been designed to be cost effective from 32 users to several hundred. Scaleability in UNICORN includes the following:

Processor Power – UNICORN is a multi-processing system and from one to four processors may be configured.

Memory Capacity – through the use of multiple memory modules and 1 Mbit and 4 Mbit DRAM technology, memory capacities from 16 to 512 MBytes are configurable.

Disc Capacity – UNICORN allows great flexibility in the number of disc drives which may be attached. Total capacity ranges from 760 MBytes to 5 GBytes in a single cabinet with capacities of up to 15 GBytes using expansion cabinets.

I/O Capability – through the use of multiple I/O cards and expansion cabinets there is extreme flexibility in the user connectivity, communications, and networking capability that may be configured.

1 The UNICORN Architecture

A systems architecture has been defined for UNICORN that includes the CPU board set, including memory, and the I/O sub-system. It uses a dual bus architecture: the industry standard VMEbus ("Versa Module Europe" – the most popular 32-bit bus in the industry) for I/O and the 64-bit ICL proprietary HSPbus (High Speed Private) for interfacing to memory.

The architecture of UNICORN embodies certain principles which are fundamental to an understanding of the architecture.

Multiprocessing – In order to allow flexibility in configuring performance to user requirements and to provide the performance which will be required through into the 1990s, an architecture which supports true symmetrical shared memory multi-processing is mandatory. Symmetrical multi-processing allows any task to be executed by any processor in the system. This gives predictable performance which is scaleable with the number of processors. UNICORN allows up to four processors to be configured.

Industry Standard I/O bus – One of the requirements for UNICORN is as an OEM platform. To allow third parties to have the option of utilising off the shelf I/O controllers, the I/O bus is an industry standard VMEbus.

Caching – The UNICORN CPUs have caches. A cache is a very high speed local memory which contains frequently accessed instructions and operands. This is necessary to allow the SPARC microprocessor to run at full speed. Also a cache is necessary to reduce the bandwidth to main memory so that multiple processors can share a common memory subsystem. Thirdly I/O transfers also need to be buffered or cached so that I/O traffic does not interfere with and slow down CPU/Memory traffic on the High Speed Private Bus.

Coherency – The use of caching on the processors and I/O is necessary for performance. This is a hardware technique for improving performance and so should be transparent to software as much as possible. Caching potentially introduces the problem of stale data when caches become out of step (incoherent or inconsistent) with main memory and each other. The architecture, is therefore designed to provide cache coherency through hardware snoop (bus watching) logic.

These fundamental design principles lead to the conceptual architecture shown in Fig. 1.

A CPU is buffered by a cache and sits on the HSPbus. Other CPUs (up to four) are buffered and interfaced in a similar way. Memory modules also sit on the HSPbus and thus are equally accessible from all processors. This creates a symmetrical shared memory multiprocessing system.

I/O controllers sit on the VMEbus and this is linked to the HSPbus via a cache; so the I/O and CPU interfaces are logically identical.

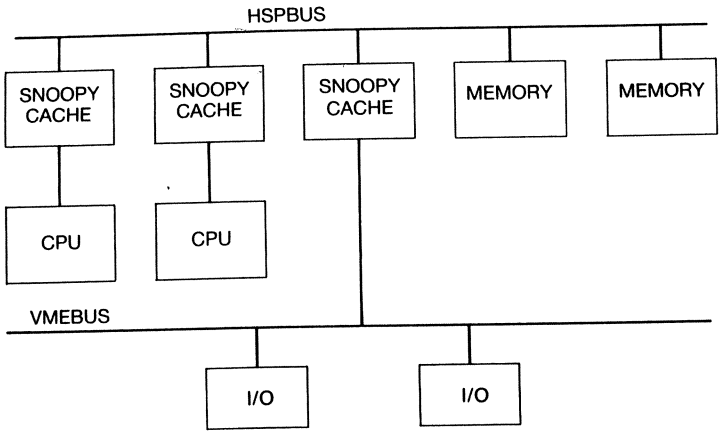


Fig. 1 UNICORN conceptual block diagram

As all modules which talk to memory are buffered by caches and interface to memory via the HSPbus, this becomes the common point for implementing cache coherency. By adding hardware snoop facilities to each and every cache, a cache can be made aware of the activities of all other caches. An algorithm can then be established to maintain cache coherency.

1.1 UNICORN implementation

This section shows how the fundamental design principles outlined in the previous section become the implementation which is UNICORN. Figure 2 shows the architecture of UNICORN where it can be seen that it is equivalent to the logical architecture already described.

The I/O caching facility appears only once in a system, so other system facilities which are required only once have been incorporated into the I/O module and it is now named the Central Services Module. An example of the facilities included are the Boot and Diagnostics functions.

The CPU modules have gained an interface to the VMEbus. The cache in the Central Services Module is only useful for VMEbus slave transfers, i.e. those transfers which are initiated by another master on the VMEbus. VMEbus master transfers, initiated by the CPU, cannot be buffered by a cache and what is more may be slow. Therefore to avoid loading the HSPbus and because these accesses are not cached, the CPU modules have a separate VMEbus master interface.

The memory modules also have gained an interface to the VMEbus. This is a slave-only interface which gives access to diagnostic and configuration information on the memory module.

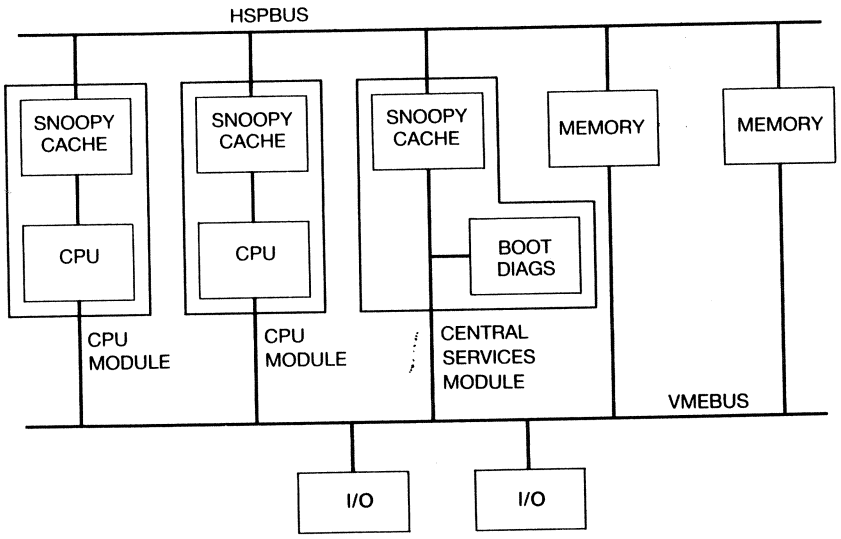


Fig. 2 UNICORN architecture

The following sections describe in more detail each of the major functional modules of the core UNICORN architecture. These are:

- CPU Module
- Memory Module
- Central Services Module

2 CPU Module

The CPU Module is a SPARC based processor module. Functionally it may be divided into several submodules. The submodules to be described in the following sections are as follows:

- Integer and Floating Point Coprocessor
- Cache
- Cache Management and Memory Management Units
- Snoop
- VMEbus Master and Slave Interfaces
- Interrupt Handler.

A block diagram of the CPU Module is given in Fig. 3 which shows the relationship of the various submodules.

2.1 Integer Unit and Floating Point Coprocessor

The CPU module design is targeted at a particular implementation of the SPARC microprocessor; the integer unit runs at 33 MHz and gives a

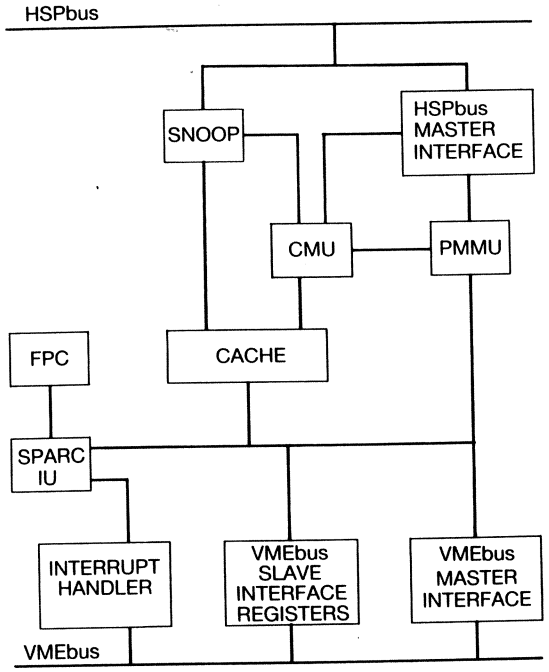


Fig. 3 CPU Module Block Diagram

performance of 15–20 mips. This is described by the vendor as a 20 mips part. The processor has associated with it a floating point chip which interfaces to the main integer unit.

The integer unit does not include memory management and so generates a 32 bit virtual address. This is extended by a 16 bit context number to distinguish one task from another.

Context 0 is the kernel context used by the operating system.

Contexts 1 through FFFE (hexadecimal) are used for user contexts.

Context FFFF (hexadecimal) is reserved for cache flushing (clearing out the contents of the cache).

2.2 Cache

The SPARC CPU generates virtual addresses and it is not possible to translate the address through an external Memory Management Unit (MMU) to access a physical cache within the cycle time of 30 ns. A virtual

cache is therefore implemented where the cache is accessed directly with virtual addresses.

The cache has the following characteristics:

- 128 Kbyte cache
- 64 bits wide for refill
- 32 byte line
- Copy back
- Cache coherency.

A copy back strategy, where memory is only updated when a line is displaced, reduces bus bandwidth over a write through strategy. Copy back is necessary because of the bus bandwidth limitations in the multi-processor case. Copy back also increases CPU performance. It does, however, increase the complexity of cache coherency. Figure 4 shows a block diagram of the cache.

The cache comprises five RAM arrays.

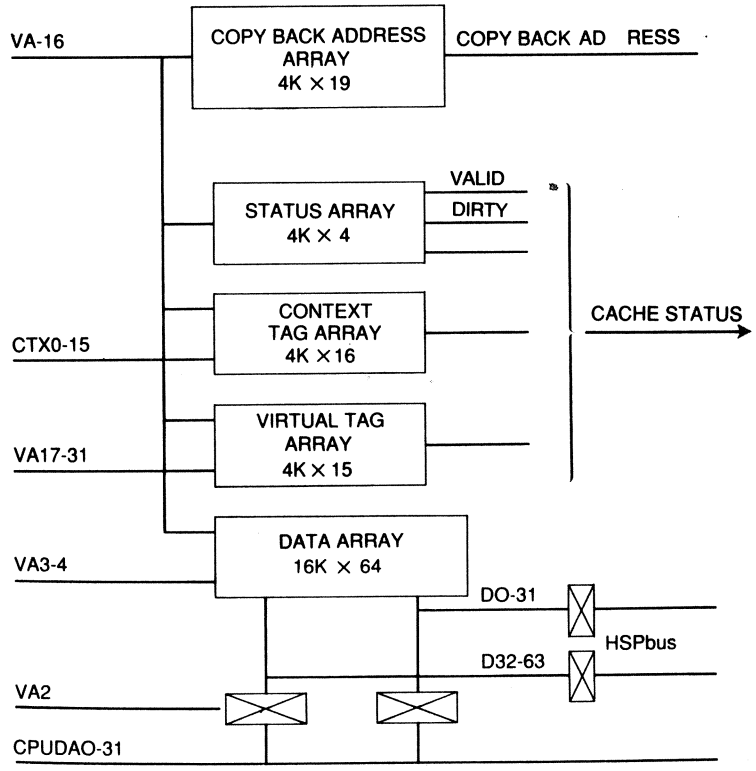


Fig. 4 CPU module virtual cache

The data array contains the data which is cached. It logically has two ports; one to the CPU chip and one to main memory. The performance of the cache is very much dependent upon the product of the miss rate and the time to refill a cache line on a miss. The miss rate is minimised by having a large cache (128 Kbytes), and large line length (32 bytes), and the refill time is minimised by having a wide path to memory, in this case 64 bits wide. The data array organisation is therefore 16K × 64. Address bits VA3–16 address the data array.

When accessing the data array, a cache hit is dependent upon:

- The virtual address tag in the tag array matching the virtual address presented by the CPU chip.
- The context number from the context array matching the context register.
- The status bits read from the status array, e.g. valid.

With a line length of 32 bytes, address bits VA5–VA16 address the tag, context, and status arrays.

Copy back means that writes go only to the cache and not immediately to main memory. Main memory is updated only when a modified cache line is flushed from the cache. Because the memory management unit can only translate addresses for the current context and the supervisor context, it is necessary to save the physical address associated with each line in the cache for use during copy backs. These are saved in the copy back address array.

The cache will need to be flushed under hardware control. On power-on reset the status RAM will be reset to make all cache lines invalid.

The cache will also need to be flushed under software control. This occurs when context numbers are to be reused. As at this point the cache may contain modified lines, it is not possible to flush the cache by resetting the valid bit in the status array otherwise the data in the modified lines will be lost. In this case the context array is reset to FFFF(Hex) and the status bits are maintained. Context FFFF(Hex) is not used by the processor and so effectively the cache has been flushed. Modified lines now in context FFFF(Hex) will be copied out to memory as they are displaced.

2.3 The Cache Coherency Protocol

Support for cache coherency is distributed through the system and resides in the cache controllers, the snoop logic, and the HSPbus protocol. A caching algorithm is defined which gives high performance with full cache coherency. This is a copy back algorithm which requires that each line in the cache can be in one of several states. Four cache line states are required. These are:

- INVALID
- SHARED NON DIRTY (SHARED)
- EXCLUSIVE NON DIRTY (PRIVATE)
- EXCLUSIVE DIRTY (MODIFIED)

The meanings of these terms are:

DIRTY	the cache line has been updated but the corresponding line in main memory has not; therefore this line must not be simply discarded but must first be copied back to main memory
SHARED	the line may be present in more than one cache
PRIVATE	the line is present in one and only one cache.

The status of a cache line will transit from state to state on both processor based activities and bus activity. Bus activity is monitored by the snoop logic described in the following section. The cache consistency protocol is shown diagramatically in Fig. 5 and is described as follows:

Processor activity

Read hit. Data is read directly from the cache and there is no state change.

Read miss. If the line is MODIFIED it is copied back to main memory and the new line is read in. All other caches capture the address of the read and

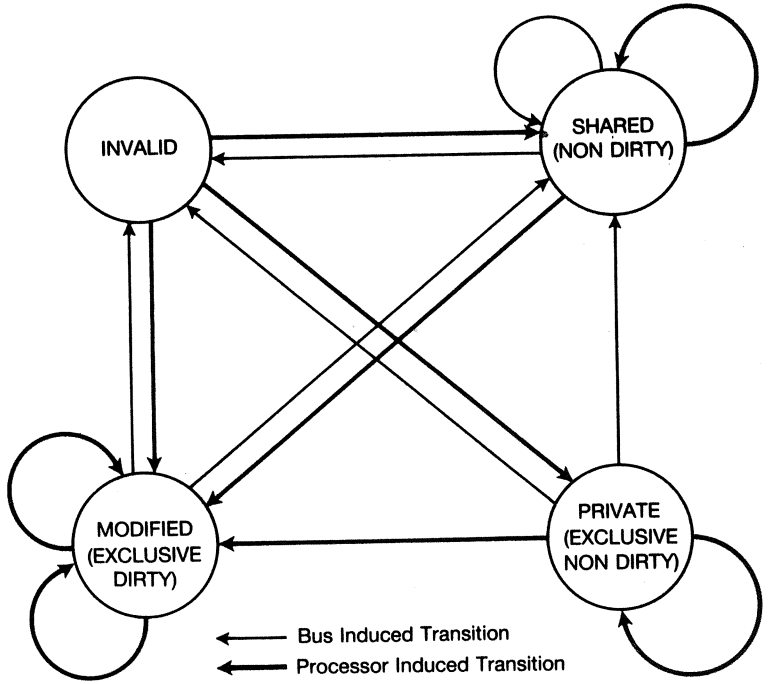


Fig. 5 UNICORN state transition diagram

check whether they have cached that line. If it is in another cache as MODIFIED the read from memory is intervened (intercepted) and the MODIFIED line is written back to memory. The original read then continues. If the line was in another cache as PRIVATE or SHARED, then they will signal the fact via an HSPbus signal and the line will transit to SHARED. The other cache line, if PRIVATE, will also transit to SHARED. If no other cache signalled that it held the line, the line becomes PRIVATE.

Write hit. If the line is PRIVATE, it is necessary to check for write permission before proceeding. If OK the line status changes to MODIFIED. If the state was already MODIFIED, the write proceeds immediately and there is no state change. If the line is SHARED then the physical address is broadcast on the HSPbus (ABI, address broadcast invalidate) to other caches holding the line. The other caches will transit to INVALID and the original cache will transit to MODIFIED.

Write miss. A write must perform a write allocate. This means that the line must be read in before the write can proceed. Therefore first follow the sequence for a read miss, then the sequence for a write hit.

Bus activity

Address Broadcast Invalidate. When an address broadcast invalidate is transmitted over the HSPbus, each snoop unit will look up the address broadcast in its cache. If it hits in the cache then that line will transit to INVALID.

Reads. The addresses of all reads are monitored by the snoop units. If the address hits in another cache and the line status is MODIFIED, then the snoop unit will intervene in the current bus transaction and copy the line back to memory. If it hits in the cache and the line was PRIVATE or SHARED then the snoop unit will signal "shared" on the HSPbus and the status of the line read and the line snooped will both transit to SHARED.

Writes. Line writes will only ever occur when a MODIFIED line is displaced from the cache. By definition a MODIFIED line can only be in one cache and so line writes will not be snooped. The addresses of non-line writes may be snooped in another cache and the status of the snooped line may be PRIVATE, SHARED, or MODIFIED. PRIVATE and SHARED lines will transit to INVALID. A snoop hit on a MODIFIED line will cause intervention and the modified line will be copied back to memory before the original write is allowed to continue.

The cache coherency algorithm may also be described by a state transition diagram. This is given in Fig. 5.

2.4 CPU Module Snoop Unit

The snoop logic will capture all physical addresses which appear on the HSPbus and determine whether the line represented by that physical address has been cached in the CPU's virtual cache.

The snoop therefore contains as tags all of the physical addresses which correspond to the virtual address tags in the CPU's cache.

Using the physical address from the bus as an index into the snoop tag gives a choice of 16 physical tags to compare the physical address with. This is because the page size is 8K and the cache is large enough to contain 16 × 8K pages (128 Kbytes). A hit occurs when a physical address on the bus matches one of the tags. In this case the state of the corresponding line in the CPU's cache must be modified (e.g. invalidated) or the shared state signalled on the bus. As a single physical memory location may map to several virtual addresses (i.e. shared memory synonyms) then there could be several hits in the snoop tag. This would be difficult to handle and so a restriction has been made such that synonyms will displace each other from the cache. There are two possible techniques to achieve this. The first is to align synonyms on 128 Kbyte boundaries, this being the cache size. Synonyms will then naturally displace one another from the cache. An alternative but more complex method is self snoop. Whenever a cache line refill takes place, the rest of the cache is snooped to check for synonyms. If a synonym is found then the entry is invalidated. Of course if it was MODIFIED it is copied back first. Either technique ensures that there is only ever one instance of a line of data in the cache and there are no synonyms.

As already described the snoop is required to match 16 tags simultaneously. In effect the snoop is a 16 way set associative cache. Normally the implementation of such a cache would be impractical in terms of the amount of hardware required. However, there is now available a 4 way associative tag RAM as a single chip. This makes a 16 way associative tag feasible with four chips.

There are two operations required of the snoop. The operation already mentioned is the snoop itself. The other operation is the loading of the snoop tag with the physical addresses corresponding to the virtual addresses in the CPU's cache, i.e. on a cache refill.

On a cache refill the snoop array is addressed by the virtual address from the processor. VA0–VA4 address the byte within the cache line and so are not used here. VA5–VA12 are used to index into the tag selecting one of 256 sets, each set containing 16 tags. VA13–VA16 are used to select the way where the physical address used in the refill will be loaded as a new tag using PA13–PA31.

On a bus snoop the physical address from the bus is captured in a register. This is now used to address the snoop array. PA0–PA4 again are not used. PA5–PA12 are used to index into the array to select a set containing 16 tags. PA13–PA31 is then compared associatively with all 16 tags.

There are 16 separate hit lines from each of the ways but, because of the restriction on synonyms, only one hit line will assert if there is a hit. A 16 to 4

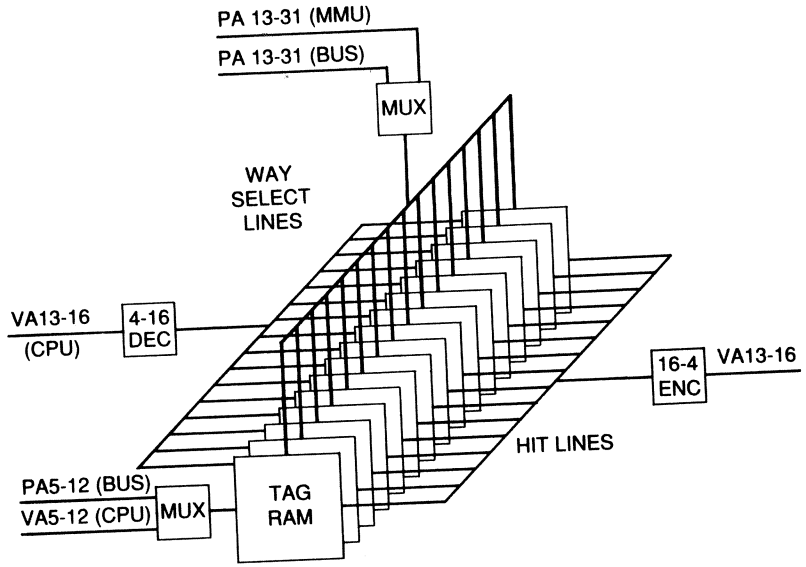


Fig. 6 CPU module bus snoop block diagram

encoder is then used to recreate the virtual address bits VA13–VA16 which together with PA5–PA12 give the location of the line in the CPU's cache which has to be modified. With a page size of 8K, PA5–PA12 are the same as VA5–VA12, i.e. they are not translated. Figure 6 shows a diagram of the snoop.

2.5 Cache Management and Memory Management Units

The purpose of these submodules is to translate the virtual address emitted by the integer unit and to process integer unit cache misses and service snoop hits.

The memory management function provides a translation buffer of 64 page table entries and has hardware controlled page table walking for replacing entries on a miss.

The Cache Management Unit is implemented as a microprogrammed state machine executing the cache coherency algorithm.

2.6 VMBEbus Master and Slave Interfaces

The VMEbus Master function is required when the CPU needs to access devices (slaves) on the VMEbus. An ASI (Address Space Indication) decode is defined for addressing the VMEbus. On decode of this ASI the address is presented directly to the VMEbus. Both the cache and the MMU are

bypassed. The VMEbus is arbitrated for and the access to the slave takes place, either sourcing data from the CPU for a write or returning data to the CPU for a read.

All VMEbus accesses are qualified by an address modifier. The address modifier presented on the VMEbus is given by the contents of an internal register. Some of the address modifier codes define the following types of access.

- 32 bit (extended) addressing
- 24 bit (standard) addressing
- 16 bit (short) addressing.

A VMEbus slave module is also required. This will provide a number of registers in a user-defined VMEbus A16 address space. This reserved space is known as the Controller Interface Table (CIT). The address of the base of these registers will be determined by the geographical address lines picked up from the backplane, i.e. the slot number. Registers may be read from the VMEbus to provide the following information.

- Board type (CPU)
- Board status (fail, etc.)
- Interrupt status.

Registers may be written to from the VMEbus to drive the following on board functions.

- Board reset
- CPU halt and restart
- to cause and mask interrupts.

2.7 Interrupt Handler

An interrupt handler is provided to handle all seven VMEbus interrupts. In a multiprocessor system all CPUs will have the VMEbus interrupt handler logic but the task of interrupt handling at any one level will only be handled by one CPU at a time. The VMEbus interrupts are combined with other interrupts for presentation to the integer unit.

3 Memory Module

The memory module resides in the HSPbus and provides 32 MBytes of Memory using 1 MBit DRAMs or 128 MBytes using 4 MBit DRAMs. The boards are capable of being depopulated to 16 MBytes and 64/32 MBytes respectively. Figure 7 shows a block diagram of the memory modules.

The memory module receives a 32 bit address and sources or sinks data on a 64 bit data bus. The memory is designed to operate in either 32 bit or 64 bit

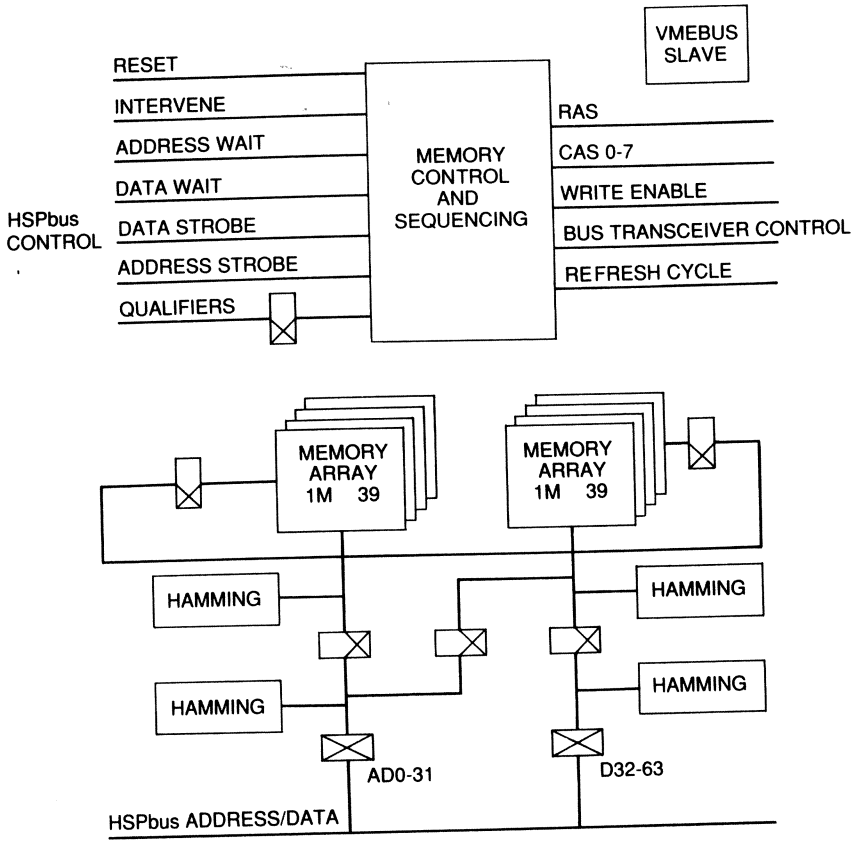


Fig. 7 UNICORN Memory Module

mode. In 32 bit mode, the data lines used are AD0–AD31. In other words the data is shifted to the least significant end of the data bus.

The memory module is capable of handling the following functions in 64 bit mode:

- Read of 64 bits, representing 8 consecutive bytes.
- Read of 2 × 64 bits, representing 16 consecutive bytes.
- Read of 4 × 64 bits representing 32 consecutive bytes.
- Read of 8 × 64 bits representing 64 consecutive bytes.
- Write of 64 bits, representing 8 consecutive bytes.
- Write of 2 × 64 bits, representing 16 consecutive bytes.
- Write of 4 × 64 bits representing 32 consecutive bytes.
- Write of 8 × 64 bits representing 64 consecutive bytes.

With writes for each 64 bit quantity 8 validity bits are presented, with the data corresponding to the bytes which are to be actually written.

The following operation is supported in 32 bit mode:

- Read of 32 bits, representing 4 consecutive bytes.
- Write of 32 bits, representing 4 consecutive bytes.

With writes for each 32 bit quantity 4 validity bits are presented, with the data corresponding to the bytes which are to be actually written.

Reads and non-line writes are capable of being aborted by the "intervene" signal on the HSPbus. This turns the transaction into a line write and data is presented to the memory from a cache other than the one which was doing the original transaction. Upon completion of the write, the original transaction continues. This feature is required for the cache coherency protocol and is known as intervention.

Memory protection is provided by Hamming checks. Writes to the memory generate Hamming check bits which are written with the data. Hamming is generated over 32 bits and so each 32 bit word has associated with it 7 Hamming bits. Reads cause the Hamming bits associated with the data to be checked. An error will cause the HSPbus signal MERR* (Memory ERRor) to be asserted. Because of the fast access time of the memory module it is not possible to correct Hamming errors on the fly. On detection of an error, therefore, the memory module performs an internal correction cycle after the event. The master initiating the transfer has to repeat the failing transaction. The retry will succeed if the error was soft and was corrected internally by the memory. If the error was hard but correctable, it is necessary to set the memory into on the fly correction mode. This slows down the access time of the memory module but allows hard correctable errors to be corrected.

Although the main interface to the memory is via the HSPbus, a VMEbus slave interface is also required for status and configuration. This will provide a number of registers in the user defined VMEbus A16 address space. These registers are used for a Controller Interface Table (CIT). The address of the base of these registers is determined by the geographical address lines picked up from the backplane, i.e. the slot number. Registers may be read from the VMEbus to provide the following information.

- Board type (Memory, size)
- Board status (Hamming failure)

Registers may be written to from the VMEbus to drive the following on board functions.

- Message register
- Memory base address
- Memory bank enable.

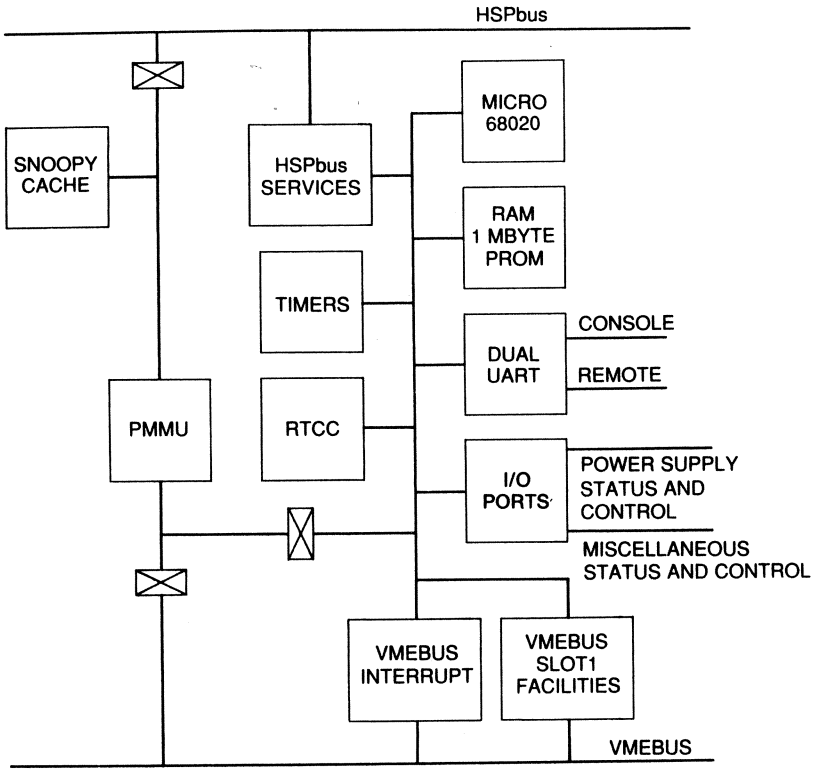


Fig. 8 Central Services Module

4 Central Services Module

The Central Services Module, as its name implies, provides services for the system. Only one CSM is configured into a system. It provides the following functionality:

- HSPbus services.
- VMEbus Slot 1 controller functions.
- VMEbus slave cache interface to memory.
- Boot and diagnostic facilities.

Figure 8 shows a block diagram of the CSM.

4.1 HSPbus Services

This functional block provides the following facilities for the HSPbus.

- Clock – this is a free running 1:1 mark/space 60 ns clock which is the bus clock from which all bus transitions are timed.

- Reset – this is the reset on the HSPB and is controlled by a bit in a register which is controlled from the diagnostic micro.
- HSPbus Error – this is a signal which is asserted if the bus had been owned for longer than 15 microseconds by a single module.

4.2 VMEbus Slot 1 Controller Function

The VMEbus System Controller functions listed below are provided:

- System Clock driver
- Serial Clock driver
- SYSRESET driver by a bit in a register controlled by the diagnostic micro
- Arbiter Module
- Bus timer module
- IACK (Interrupt ACKnowledge) daisy chain driver.

4.3 VMEbus Slave Cache Interface to Memory

This functional block provides a cache for transfers initiated by the VMEbus. Unlike the CPU caches, the VMEbus cache is a physical cache. However, it does conform to the cache coherency rules established for the system. For efficiency and maximum bandwidth, block mode transfer is used on the VMEbus where possible. To avoid the VMEbus I/O controller having to take notice of page boundaries, an MMU is included between the VMEbus and the I/O cache. I/Os are then allocated segments which link into a process's page tables.

The operation of the I/O cache is as follows:

Reads All reads will cause a line to be allocated in the cache and subsequent reads within that line will come directly from the cache.

Writes Writes follow a rather more complicated procedure dependent upon whether the write is a block transfer and where the transfer begins and ends relative to line boundaries.

The first write of a non block mode write will cause write allocation, i.e. the relevant line will be read into the cache. If the line is SHARED, an address broadcast invalidate will be sent over the HSPbus to establish the line as PRIVATE and the line will be modified. Subsequent writes to that line take place without HSPbus action.

For a block mode write, if the write starts not on a line boundary, then a line in the cache is write allocated and the action is as in the previous paragraph.

If the block mode write begins on a line boundary then the write takes place to the cache line. The previous contents of the line are flushed to memory if the line status was MODIFIED. Subsequent writes also go to the cache. If

the block mode write continues to the end of the line then an address broadcast invalidate is sent over the HSPbus and the line status is set to MODIFIED. Up to that point, the line status was set to a special reserved value indicating this condition. An address broadcast invalidate is sent because the line may be SHARED.

If the block mode write ends before the end of the line, the remainder of the line is read in from memory. If the line was read in as SHARED, an address broadcast invalidate is sent over the HSPbus before the line state is then set to MODIFIED.

Snooping The I/O cache must maintain cache coherency with all other caches in the system and so snooping is required. Note that being a physical cache, snooping uses the physical address from the bus directly as a cache index.

4.4 Cache Configuration

The I/O cache size is 64 Kbytes containing 2K lines each of 32 bytes. As the addressing characteristics of I/O are different from that of a CPU, the address bits used for accessing the cache are ordered differently. Address bits used for indexing the cache are PA0–4 and PA13–23. Address bits PA5–12 and PA24–31 are used as tags. This arrangement has the characteristics that an I/O in a single page will use only one cache line.

4.5 Boot and Diagnostic facilities

These are provided by a microprocessor controller subsystem resident on the CSM. As this will be involved in power-on diagnostics, power supply sequencing and remote access, this particular subsystem will be powered separately from the rest of the system. The following functional units are provided in the Boot and Diagnostics subsystem:

- Microprocessor
- RAM
- PROM
- Local console interface
- Remote console interface
- Real time clock, calendar (battery backed)
- Timers
- VMEbus interrupter.

Power-on sequencing will be performed by the Boot and Diagnostics subsystem upon command from the control panel (on/off switch) or the remote console (remote access). It will check that all power rails are in specification before moving to the next step.

The core configuration is then established by groping the system, given that

the CFUs and memories have configuration information accessible by slot position over the VMEbus.

Next the Central Services Module is tested, followed by the memory modules. A CPU test program will then be downloaded into memory and a CPU released to execute it. In a multi-CPU system, the CPUs are tested one at a time.

Once all basic boards have checked out OK, a boot sequence is downloaded to memory and control is passed to a CPU.

Note that because of the control that the Boot and Diagnostics part of the CSM has over the system, it is not necessary for the CPUs to have boot PROMs.

5 I/O Controllers

I/O for the UNICORN system is provided by I/O controllers which sit on the VMEbus. These are standard VMEbus cards.

5.1 VMEbus Microlan II Interface

Microlan II is an ICL proprietary two wire multi-dropped interface which is used to connect Microlan II based terminals and workstations.

5.2 VMEbus Communications Controllers

Two VMEbus communications controllers are available.

One provides four independent full duplex V24 communications channels which operate at speeds up to 19·2 Kbps.

The other provides two independent channels operating at speeds up to 64 Kbps.

5.3 VMEbus Ethernet Controller

This is a single channel controller which provides UNICORN with the facilities to support OSLAN 100, 200, 300, and 500.

5.4 VMEbus SCSI Controller

This is a two channel controller which provides the ability to connect discs (magnetic and optical), and tapes (and, potentially, high speed printers) via the industry standard SCSI interface.

5.5 VMEbus Asynchronous Communications Controller

This is a 16 channel controller which provides the ability to connect VDUs, printers, and modems via the industry standard RS232 (CCITT V24) interface at speeds up to 38·4 Kbps.

5.6 VMEbus Repeater

The VMEbus repeater provides the means to expand the VMEbus beyond the 20 slots provided in the main cabinet card cage. The repeater comprises two VMEbus cards which are linked together by a cable. One resides in the main card cage and the other sits in the card cage in an expansion cabinet.

6 Summary

UNICORN provides a very flexible architecture to support its role as a multi-user UNIX platform.

The hardware allows great flexibility in configuration allowing cost effective solutions to be realised from 32 users to several hundred users.

The high performance of the SPARC microprocessor is supported by high performance caches with total hardware cache coherency through the system. This both improves performance and relieves the operating system software of the burden of maintaining cache coherency.

DRS6000 (UNICORN) software: an overview

T.M. Cole
Advanced Servers Product Centre, ICL Office Systems, Bracknell, Berkshire

Abstract

This paper gives a very brief overview of the version of UNIX which has been ported to the UNICORN. It also describes, in more detail, the extensions which have been made for UNICORN. In particular, the "extended streams" facility is described. This provides an environment which can be ported relatively easily to I/O Controller boards (IOCs) which can then support standard UNIX streams modules. This gives much greater flexibility in system configuration, as some facilities (embodied in streams modules) can be relegated to IOCs at a very low implementation cost.

Introduction

The UNICORN will support AT&T's[1] UNIX[2] System V Version 4 in its SPARC[3] ABI (Application Binary Interface) form – see [Ref. 1, 2, 3]. UNIX System V Version 4 incorporates the facilities of UNIX System V Version 3.2 and the Microsoft XENIX[4] system, and some of the facilities of the Berkeley BSD[5] system and Sun Microsystems' SunOS[6]. This will conform to the POSIX[7] (Portable Operating System Interface for Computer Environments) and X/OPEN[8] standards – see [Ref. 4, 5]. The generic source is supplied to us by AT&T, in a form as implemented on an AT&T 3B2 series computer, and we "port" it to the UNICORN and merge in ICL's "value added" components (e.g. Microlan support). This paper describes the software as it applies to this port. A general description of the software architecture of UNIX is available in [Ref. 6], for instance.

The "Porting" Task

The porting task involves taking the generic source and reworking the machine-specific parts to conform to the target machine – in this case the UNICORN. This includes removing all the AT&T 3B2-unique programs and interfaces, and modifying or rewriting others – for example the cartridge tape programs – where UNICORN equivalent hardware exists. The largest part of this task concerns the peripheral access, although a substantial amount of effort also has to be devoted to memory management and the bootstrap process.

The UNICORN hardware is described briefly in an accompanying paper in this issue. [Ref. 7]. There are three classes of hardware component in the UNICORN which are capable of executing code. The CPU (Central Processing Unit) is the only one concerned with executing application programs, as well as being responsible for the kernel functions: it does not have direct access to peripheral devices. The CSM (Central Services Module) and IOCs (Input/Output Controllers) are the only components with direct access to peripheral devices. The CSM is responsible for the system console device and an auxiliary RS232 port intended for use for diagnostic purposes. The CSM is also responsible for starting the system and for monitoring the health of the system, for example detecting cabinet overtemperature. The IOCs are responsible for all the other peripheral devices. Only the CSM and CPU have direct access to main memory: the IOCs are required to access main memory via the CSM.

The division of responsibility among the computational units requires coordination. This is achieved by exchanging messages over a bus which is accessible to all. This bus is called the IObus and is an implementation of the standard VMEbus [Ref. 8]: the "VME" here may stand for "Versa Module European", in which case the "Versa" bit refers to a Motorola proprietary bus – but there are various opinions about this.

Peripherals are accessed by the UNIX kernel via device drivers. There is a standard interface between the drivers and the kernel known as the DKI, Driver-Kernel Interface [Ref. 9]. There are three types of device driver, known respectively as **block** (mainly for disc access), **character** (also known as **raw** or **physical** access when applied to disc or tape), and **streams** [Ref. 10]. Of these, by far the largest variety of usage goes to the streams drivers, as these support the communications facilities which are central to UNICORN's departmental systems role.

Streams were introduced in UNIX System V Version 3 kernel to enable communications protocols to be modularised. [Ref. 11] defines as follows:

"**STREAMS** A set of kernel mechanisms that support the development of network services and data communications **drivers**. It defines interface standards for character input/output within the kernel and between the kernel and user level processes. The STREAMS mechanism is composed of utility routines, kernel facilities and a set of data structures."

The OSI seven layer model, for example, lends itself to modularisation, the various layers providing the initial basis for defining the functions and interfaces of the component modules. The interfaces between the modules are expressed in a standardised (streams) message passing form. This standard enables them to be interconnected in any rational sequence at run time, without the need for further recompilation or linking. There is a "streams" scheduler in the kernel which causes the various streams modules to be executed from time to time as required, to process their input messages and

modify them or perhaps generate new messages which are output to the next module in the stream.

The UNICORN extends this streams environment to include the CSM and IOCs so that communications modules may be IOC, CSM or CPU resident without needing to rework the modules themselves, or any abridging of the former flexibility of interconnection. If versions of the modules are available ready-linked for the IOC, CSM or CPU environments, the choice of stream module location can be left to run time. This extended environment is referred to as "Extended Streams" in the context of the UNICORN port.

The porting tasks associated with the non streams drivers are fairly standard, and will not be discussed here.

Extended Streams on the UNICORN

This scheme is realised with:

- a driver which can pass messages along the IObus
- a miniature kernel environment on each IOC sufficient to supply all the needs of streams modules (qv DKI [Ref. 9])
- a set of "stub" modules to represent, in the kernel environment in main memory, those modules absent from that environment.

The "stub" modules are known collectively as **relays**, as they transparently "relay" messages across the IObus. The miniature kernel environment on each IOC is known as the **ISMkernel** (Intelligent SubModule **kernel**).

UNIX streams are described in [Ref. 10], but the following very brief and simplified overview may give a sufficient background against which to present the idea of Extended Streams.

A stream consists of a stream head (which mediates between the kernel environment and the user application); optionally followed by some streams modules; and terminated by a streams driver. The stream is constructed when the driver is first opened by the user application: this links the driver to the stream head and then proceeds to "push" the modules onto the stream immediately below the stream head (see Fig. 1).

The UNICORN Extended Streams could implement this stream where, say, module B and the driver C were IOC-resident, by substituting relays for B and C in the kernel resident stream (see Fig. 2). The relays on the CPU pass any messages they receive to the IOC relay, and pass upstream any messages they receive from the IOC relay. The link from relay C remains dormant until B is "popped" from the stream, as messages from the IOC are passed straight to relay B, and relay B receives any messages coming downstream before relay C.

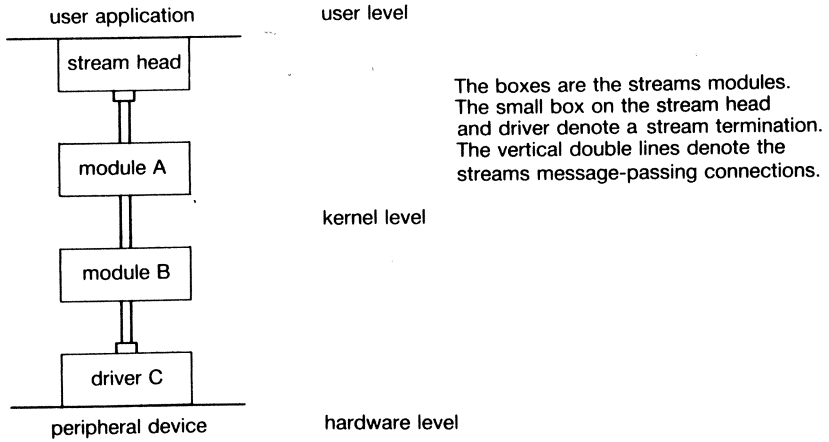


Fig. 1

Our goal was to avoid making any changes to the generic UNIX code supplied to us. The use of relays achieves this because the generic UNIX code can never detect the substitution. Each IOC-resident module or driver has to supply a corresponding relay which is able to locate "its" IOC and pass the IOC address to the relay support code. This structure could be advantageous as it is generally true that messages are more frequent at the lower levels of the communications protocol stacks. Hence the lower levels may be devolved to multiple instances of particular types of IOC.

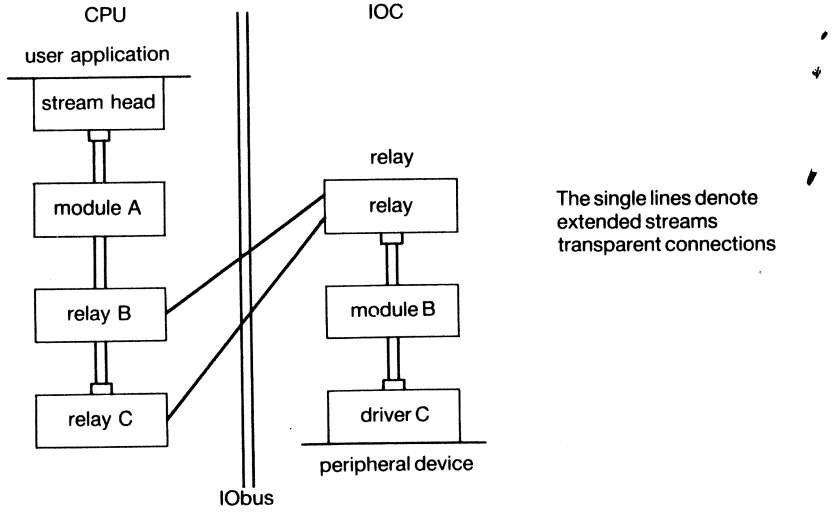


Fig. 2

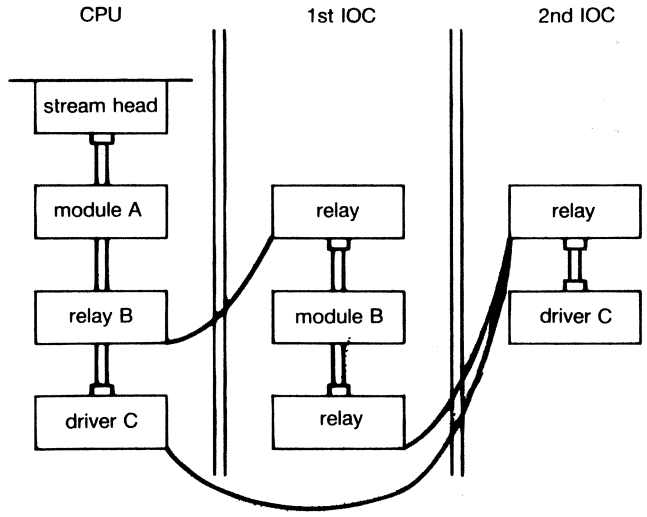


Fig. 3

In the UNICORN, IOCs can communicate with each other on the IObus independently of the CPU. This allows a stream to be extended across more than one IOC (Fig. 3). This is not very useful in its own right, but can be exploited when streams multiplexors are involved (see below).

Some streams drivers are used to multiplex or demultiplex other streams connections. In this case, a single "upper" stream terminates on reaching the multiplexor, which acts as the origin for one or more streams on the "lower" side. This is constructed at run time out of a collection of streams which have already been opened and set up (Fig. 4), one of which is the stream to the multiplexor. The streams are linked to the multiplexor, one by one, until the structure is complete (Fig. 5). The stream heads which belonged to the "lower" streams are now redundant, and would be freed by the application. If modules A1 and A2 were IOC-resident, a multiplexor could be used to "drive" more than one IOC (see Fig. 6). The relay on the first IOC which was used to link relay A1 to the module A1 has become redundant in a similar way to the way the stream head would have become redundant had the stream been CPU-resident.

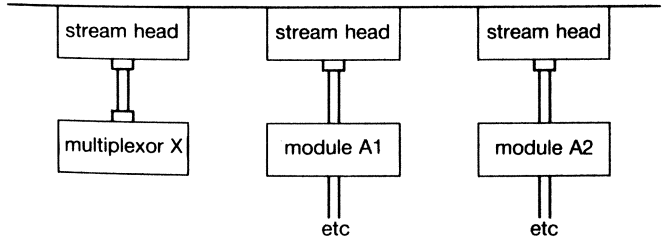


Fig. 4

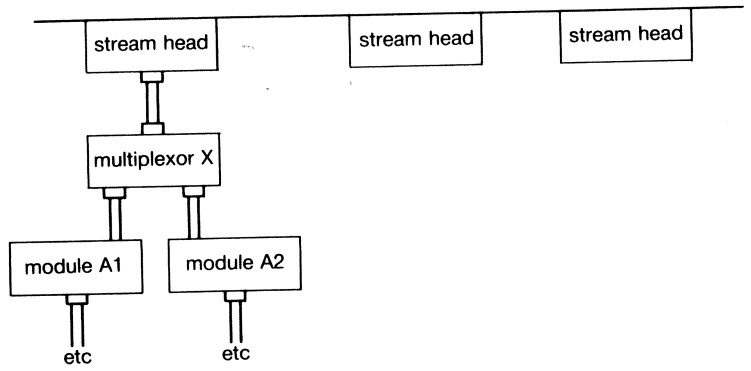


Fig. 5

However, this logical structure could also be built up as follows, with the multiplexor itself placed on one of the IOCs (see Fig. 7). In this case, the links from relays A1 and A2 to their respective IOCs become dormant until the multiplexor is dismantled, as the relay X intercepts all messages coming downstream and dispatches them to the IOC relay. Similarly, all messages from modules A1 and A2 flow to the multiplexor on the first IOC, and then via the multiplexor relay. This form may be advantageous if the multiplexor is capable of receiving messages from, say, module A1 and sending them on to module A2. Such a flow would never directly concern the CPU in this configuration.

The extended streams code is structured into layers to facilitate porting to other systems and IOCs, and also to allow for non-streams usage (see Fig. 8). There are various sub-systems which exchange messages, and are classed as different "services" within this layered structure. Hence, the extended streams use becomes the "extended streams service". Some functions required by the ISMkernel environment (e.g. passing messages to be printed on the system console) are provided by a "kernel utilities" service.

At the first release, the Microlan IOC and CSM will be fully implemented to

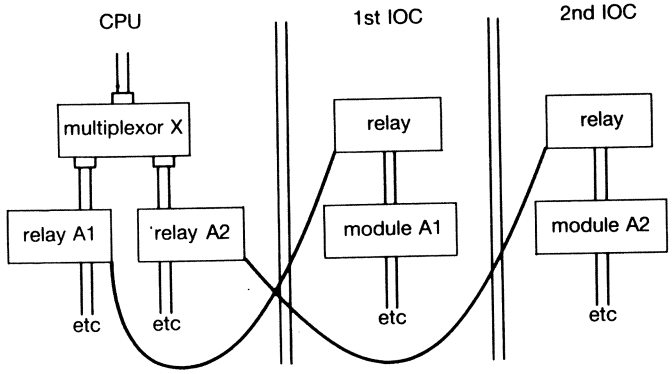


Fig. 6

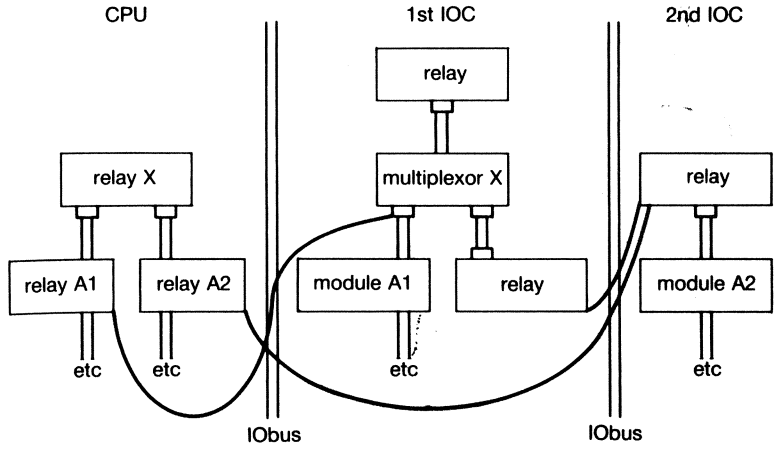


Fig. 7

this plan. The X25 IOC will interface some already implemented non-streamed software to the IObus message passing interface so that the CPU-resident parts of the protocol can be implemented as standard streams modules. In the future, it is intended to implement a high performance Ethernet[9] IOC to this scheme. Once established, the only rework required to introduce a new IOC should be to reimplement the IOC-dependent part of the ISMkernel.

Other aspects of the UNICORN software port

All access to main memory from the CSM or IOCs is made via a virtual-to-

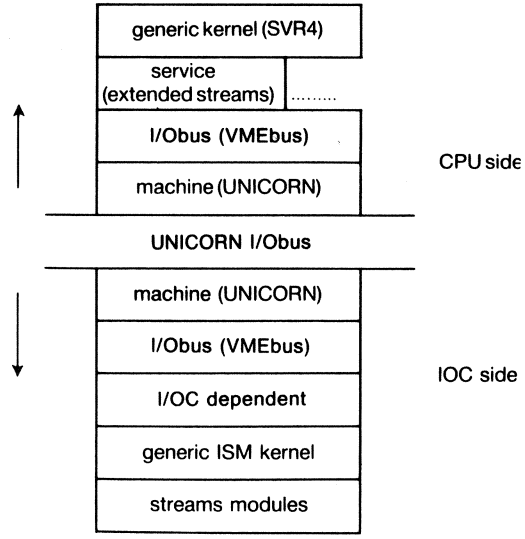


Fig. 8

physical address translation on the CSM board. This relieves the IOCs and CSM from having to "know" anything about the UNICORN memory management. It also removes the need for the CSM and IOCs to support "scatter/gather" DMA. The addresses passed to IOCs are virtual addresses within an address space which is reserved for i/o. Buffers are mapped into this space as required. Kernel-resident buffers are permanently mapped in. Application program-resident buffers are mapped in as required. The first 16 Mbytes of this i/o address space are reserved for buffers which are to be accessed by IOCs with only 24-bit addressing capability. These buffers can nevertheless exist anywhere within the HSPbus physical main memory (128 Mbytes maximum at first release).

The CSM is used to control the bootstrap process, including running establishment checks on the CPU and memory boards. In this way, it is not necessary to include any PROM code on the CPU boards. The CSM can also be used to run ETS (*E*ngineering *T*est *S*oftware) which can be controlled remotely if required.

Conclusion

The Extended Streams facility will allow more flexibility in configuring UNICORN systems, especially where large amounts of communications are to be supported. The modules, multiplexors and drivers which implement the communications protocols may be positioned in IOCs or in the CPU to optimise the system performance. Future IOCs and communications facilities may be introduced with a minimum of rework, without detracting from this flexibility of configuration.

End Notes

[1] AT&T is a registered trademark of AT&T in the USA and other countries
[2] UNIX is a registered trademark of AT&T in the USA and other countries
[3] SPARC is a trademark of Sun Microsystems Inc.
[4] XENIX is a registered trademark of Microsoft Inc.
[5] BSD is a trademark of the University of California at Berkeley
[6] SunOS is a trademark of Sun Microsystems Inc.
[7] POSIX is a trademark of the Institute of Electrical and Electronic Engineers Inc.
[8] X/OPEN is a trademark of the X/OPEN Company Ltd.
[9] Ethernet is a trademark of Xerox Corporation

References

1 'AT&T System V Interface Definition', Issue 3 – SVID89
2 'AT&T System V Application binary Interface'
3 'AT&T SPARC Processor Supplement'
4 IEEE standard 1000.3 – 1988
5 'X/OPEN portability guide', Issue 3
6 BACH, M.J.: 'The Design of the UNIX Operating System'. Prentice Hall, ISBN 0-13-201799-7
7 POSKITT, G.: 'The UNICORN Architecture'. ICL Tech. J. 6, 4 November 1989
8 IEEE standard 1014: 'VMEbus Specification Revision C1 Oct 1985'. PRINTEX Publishing Inc.
9 'AT&T System V Device Driver Interface/Driver-Kernel Interface'
10 'AT&T System V Streams Primer'
11 'AT&T System V Programmer's Reference Manual'

Electromechanical Design of DRS6000 (UNICORN)

Roger Pullen
Advanced Servers Product Centre, ICL Office Systems, Bracknell, Berks.

Abstract

The paper gives the basic principles underlying the physical design of UNICORN and short accounts of the cabinet design and construction and of the equipment modules built into it: logic rack, disc units, cooling system, power supply and internal interconnections. It gives also the national and international standards to which UNICORN conforms.

1 Basic Principles and Strategy

The electro-mechanical development process for UNICORN has required a wide range of activities and called upon many diverse skills to produce the end product. From the designers point of view, the goal was to produce a simple cost-effective solution to a set of defined requirements. For UNICORN, these were:

- Small floor-standing, desk-high cabinet
- Able to operate in an office environment
- Aesthetic styling – part of a family of office products
- Single design to cover all international requirements
- Must meet all statutory safety requirements worldwide
- Must be able to use existing production line facilities and more specifically the Ashton Mercury production line
- Easy to install
- Minimum time to repair and service
- Easy to re-configure.

The strategy we adopted in setting out to meet these requirements was:

- To design a simple unit to cover a wide range of system configurations and avoid costly cabinet swap-outs in the field
- To consist of simple fabricated frame and cover construction
- To design for minimum number of mechanical fixings
- To create a flexible design catering for either inhouse, subcontract or overseas manufacture
- To house industry-standard units, i.e. 19″ rack type peripherals
- To create flexible cabling system and eliminate connector bulkheads

- To maximise the use of computer aided tools throughout the design process
- To use a common power supply throughout the product range to reduce spares holding. This should also cater for international mains voltages and frequencies
- To design in quality through continuous validation throughout development
- To offer an uninterruptable power supply interface
- To apply Value Engineering techniques throughout development.

Implementation

Past experience in the design of similar products has highlighted the advantages of using a simple open frame construction to house bolt-in equipment. This type of design enables future upgrades to be incorporated with the minimum of alteration to the basic cabinet. UNICORN has followed this principal throughout.

The UNICORN cabinet has been designed to provide a robust construction, with a pleasing appearance that blends agreeably with most environments, and offers a family image with other ICL products. The simple, but ingenious, construction gives a wide choice of mounting positions for individual bolt-in modules or alternatively standard 19″ rack mounted equipment. Additional cabinets can be abutted to expand the system. The need to obtain relevant company and international approvals/certification has also been considered. Figure 1 shows the basic construction and the major items of equipment.

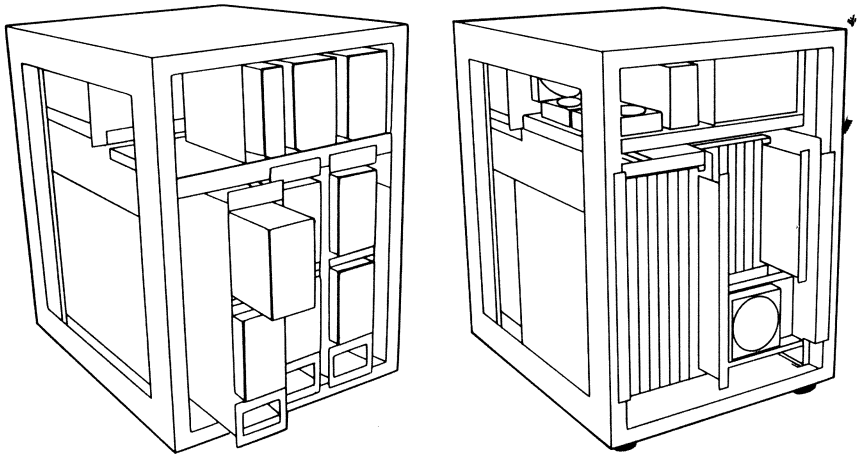


Fig. 1

2 The UNICORN Cabinet

2.1 Cabinet construction

Frame The cabinet frame is constructed of six pre-plated steel (Zintec) members – base, top and uprights spot-welded together to form a simple, robust, open framework which provides:

- 19″ compatible front and rear access
- Four adjustable feet to level the cabinet on uneven floors
- Four castors for mobility during installation
- Versatile design for mounting of equipment
- Flexible system for securing and earthing interface cables without the restriction of bulkhead connectors.

The frame has no costly secondary finishing operation as it is totally obscured by the decorative (external) covers.

External covers The appearance of the cabinet is designed to conform to the ICL product styling requirements to ensure visual compatibility with other ICL equipment, e.g. workstations.

The covers have been designed to enable them to be safely stored until the cabinet has been fully assembled and tested, i.e. covers are fitted just prior to packaging the equipment for delivery.

The painted steel top and side covers are secured in position by integral metal clips which engage in rectangular cut-outs on the frame. This has two advantages; firstly, there is no visual retention method, and secondly, the metal clips provide excellent earthing continuity for safety and radio frequency interference (RFI) shielding.

Front and rear door panels are moulded in Noryl structural foam. This material is a standard engineering plastic with a foaming agent added during the moulding process, which produces a fine, rigid honeycomb construction with a good impact strength. The cooling vents are an integral part of the moulding on both panels, but two secondary mouldings are added to provide access flaps to the exchangeable peripheral devices and the panel securing lock. See Fig. 2.

2.2 Cabinet Equipment Modules

There are five basic modules:

- Logic rack
- Disc modules
- Power supply shelf
- Fan shelf
- Control Panel

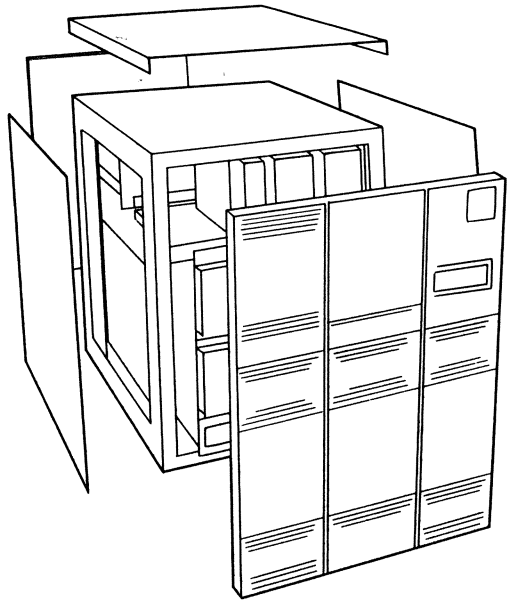


Fig. 2

Except for the logic rack, all modules are easily accessed from either the front or rear of the machine.

All of the modules except the logic rack are secured by either two or four fixings, with electrical connections plugable for ease of fitting and removal, i.e. for servicing.

2.2.1 Logic rack: The logic rack has been designed to support the industry-standard VME racking specification, and accepts standard 366·7 mm × 280 mm (9U) 9 off and 233·35 mm × 160 mm (6U) 11 off printed circuit boards plugged into a common backplane. The different depths of PCBs resulted in ICL-designed metalwork to enable advantage to be taken of VME standard proprietary parts – extrusions, tapped strips, insulators, guides and fixings. See Fig. 3.

A unique feature of the logic rack is the provision to allow small (I/O) PCBs (366·7 mm × 100 mm) to be mounted directly behind the existing PCBs; these mate with reverse DIN connectors on the backplane. Extensions to the rack side panels allow additional extrusions to be mounted both top and bottom, which in turn form the mountings for the card guides for the (I/O) PCBs.

The 6U PCB area of the rack accepts either ICL-designed or externally-sourced PCBs, all fitted with standard VME front panels.

The 9U PCBs and front panels are totally ICL-designed. Particular attention

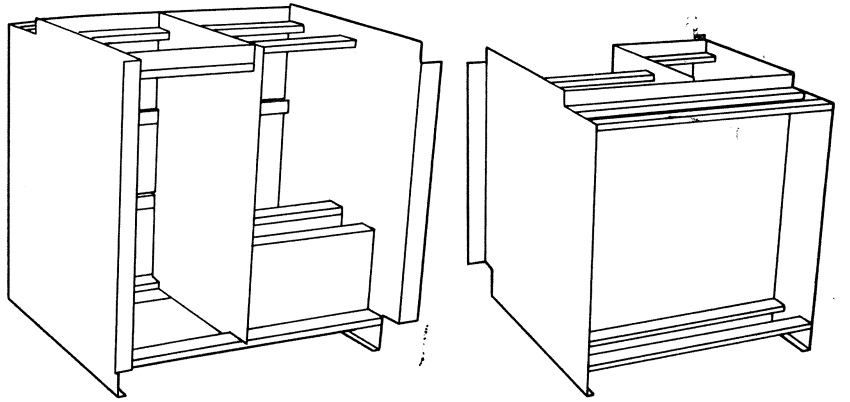


Fig. 3

has been given to the design of the 9U PCB front panels so that a good EMC seal is achieved between the rack side panel and each PCB front panel. The 9U PCBs are also fitted with a purpose-designed board stiffener to assist with the alignment of the three connectors fitted to the backplane. See Fig. 4.

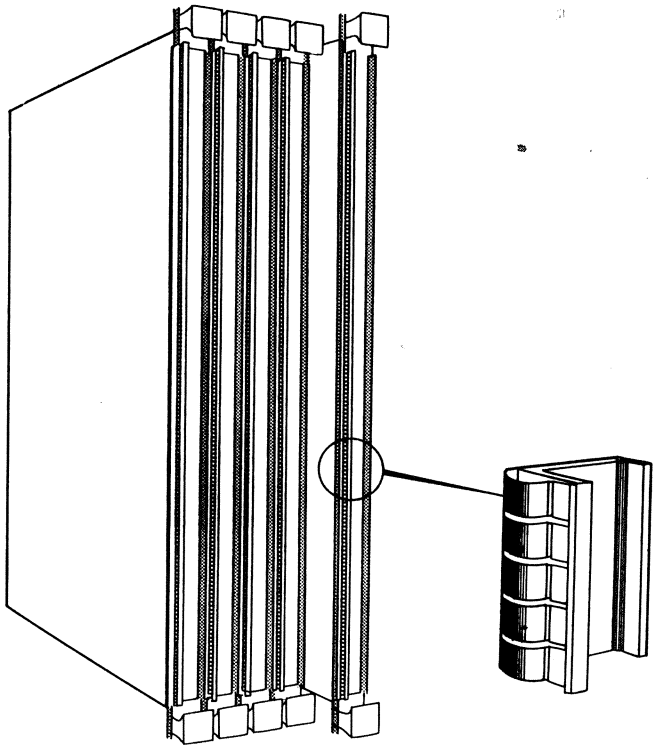


Fig. 4

2.2.2 Disc modules: The basic cabinet houses up to a maximum of nine 5·25″ peripheral devices, which may be a combination of fixed discs, optical discs, floppy discs and cartridge magnetic tape drives.

The exchangeable media devices are mounted at the top of the machine, directly behind the hinged flap on the front moulding, so that the operator can load/exchange media.

Fixed disc drives are mounted within bolt-in modules, which provide a flexible system of enhancement that avoids modification to the basic cabinet. Each module kit consists of a housing, disc drive, mounting brackets, power and signal cables, 'T' connectors and terminators to enable up to two drives to be fitted and connected into the basic system. The 'T' connectors enable the signal cables to be daisy-chained, and a terminator to be fitted.

Under consideration for the future are plug-in drive adaptors to eliminate the power and signal cables completely; this would reduce the time for installation and maintenance and increase reliability. See Fig. 5.

2.2.3 Cooling system: Basically, each heat-generating module is individually cooled (i.e. each module draws in air at ambient temperature). Cooling air is drawn in through the lower vents in the front and rear covers and is exhausted through the upper vents in the rear covers.

The fan module provides cooling for the 9U PCBs as well as the three

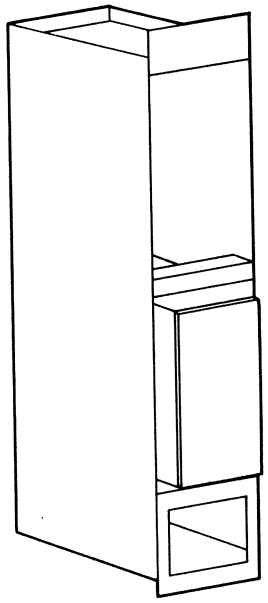


Fig. 5

peripheral devices mounted at the top of the cabinet. The 6U PCBs are cooled by a separate fan mounted at the front of the logic rack, which exhausts into a common plenum area within the fan module. Each disc module contains its own cooling fan.

Thermostats mounted above the logic rack monitor temperature within the machine and in the event of a system over-heat will instigate a controlled system shut down thus avoiding prolonged heat damage to sensitive devices.

2.2.4 Power supply unit: The power system for UNICORN was conceived as a single industry-standard package which derives all the necessary DC rails from the AC mains. Due to space constraints within the cabinet, switch mode techniques are necessary to achieve the required power conversion in the size, and weight, constraints imposed.

Sizing of the load produces a required power output of just under 1500 watts. The relatively high efficiency of switch mode power supplies (>75%) means a manageable RMS current for normal domestic/office mains outlets within the UK. However, UNICORN is targeted at a worldwide market and a power system that accepts a range of input voltages and frequencies offers considerable manufacturing and servicing advantage. The big stumbling block in achieving this aim is the low single phase voltage (120 V) in North America, switchmode power supplies being tolerant to the frequency change between 50 and 60 Hertz.

A survey of electrical installations in North America reveals that phase-to-phase outlets are commonplace; a nominal voltage of 208 volts is available and applying tolerances to this and to the voltage ranges encountered in the UK, Mainland Europe, the Middle and Far East and Australia gives a working range of 180–264 volts, either 50 or 60 Hertz.

In order to give the UNICORN system a means of resilience to external AC power failures, an Uninterruptable Power Supply (UPS) is provided as an option. UPSs re-generate the AC voltage waveform from a static inverter. Usually, unless considerably over-rated for the job, UPSs are unable to supply large peak currents. But switchmode power supplies, by the very nature of their design, gave high repetitive peak currents, up to four times the RMS current, due to the repeated re-charging of the rectified mains reservoir capacitors every half cycle of AC input, and for the same reasons they have a high inrush current at switch on.

Both these characteristics are particularly unkind to a UPS. A market survey of UK power supply manufacturers showed that a number of the leaders were aware of this problem, and one manufacturer had a 1500 watt multi-output power supply in production with a front end "correcting" circuit which enabled a very near sinusoidal current to be drawn from the mains over the whole period of the AC cycle.

The overall strategy for UNICORN includes a facility for remote diagnos-

tics. This facility is included in a central services module (CSM) within the system. This module is powered separately from the main system, and allows diagnostics to be run from a remote user port. As it is impossible to perform any sort of diagnostics on a system with a power supply partly or wholly shutdown because of a fault, a small (50 watt) power supply is provided to power a subset of the interface logic on the central services module and its associated cooling fan, and thus make it independent of the main system power supply.

2.2.5 Interconnect and Backplane:

Interconnect The overall philosophy adopted for UNICORN is to employ bus bars and/or woven pre-formed cableforms as much as possible. This enables whole assemblies to be manufactured off-line and simply installed in the cabinet, with the aid of Velcro, or similar, at final assembly.

The system signal loom is such an assembly and carries cabinet housekeeping, control information and auxiliary power. It consists of discrete wires woven together and pre-formed into the required shape, with all the ends having correct terminations fitted.

The five volt feeds to the backplane are similarly treated, and employ a moulded bus bar assembly which bolts directly onto the main power supply terminals. From this moulding, separate cables extend to the backplane in pairs, the pairs being separated by weaving techniques.

To interconnect the data interfaces of the several in-cabinet small computer system interface (SCSI) peripherals, a novel approach was adopted. A kit of three parts, namely a woven cable, a moulded 'T' junction, and a main SCSI terminator, are used to perform all interconnect functions. Each disc is fitted with a 'T' junction, which provides an additional two connectors at the rear of the drive. The first connector receives the incoming SCSI cable from the adaptor, while the second either daisy-chains to the next device in the chain or is fitted with a SCSI line terminator. For field upgrades, the terminator is removed from the current last disc and plugged into the 'T' junction on the new disc; a daisy-chain linker cable then connects the new disc into the system by plugging into the position previously occupied by the terminator. This arrangement allows very simple upgrading of an installation. See Fig. 6.

The DC power to the SCSI peripheral is routed from the power supply by bus bars formed of insulated copper laminates, the individual connections to the disc are effected by woven jumper cables. Where cables interfacing to external peripheral devices enter the cabinet a "window" is included in their construction to give all-round exposure to the overall braid screen of the cable. This screen "window" is placed under a clamping bar, which is secured to an integral part of the chassis, and thus effects a very positive method of strain relief while achieving the near 360 screen termination required to obtain a high level of resistance to static discharge and effective RF screening. This method is employed on all external interface cables.

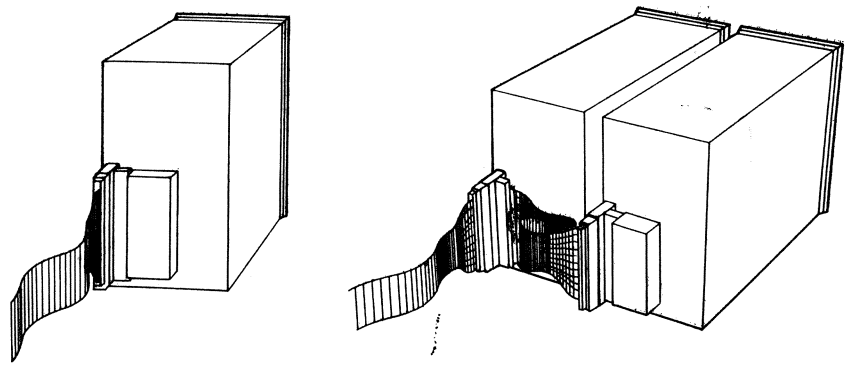


Fig. 6

Backplane The UNICORN backplane carries two bus systems:

- VMEbus
- A private bus of proprietary design (HSPbus)

It accepts up to a total of 20 PCBs. The first nine slots take triple-high (9U) Euro-card PCBs which interface to both bus systems. The remaining 11 slots take double-high (6U) PCBs conforming to the accepted VME standards, and thus only interface to the VMEbus.

Considerable thought has been given to the physical layout of the backplane, in two areas:

- The signal interface connector of the VME card slots
- DC power distribution.

For the former case, an interface connector (reverse DIN) is provided on the rear of the backplane to allow an interface card to plug up into the same slot as the VME card. It is envisaged that these cards be used for custom interfacing and providing extra facilities not normally obtained with a standard VME card. The adoption of this policy dictates the use of a backplane with "press fit" connectors. This is a new process to ICL, but one that has been widely used in the telecommunications industry for some years, so the technical risk is felt to be minimal.

Regarding DC power distribution the need to minimise the inductance of the 5 volt distribution is judged as vital, to ensure there is no "current starvation" when very high speed clocks are employed and/or large banks of memory chips are refreshed simultaneously.

To achieve this, four 2 oz copper power planes are employed, laid up alternately, to improve high frequency decoupling, and 10 000 μF of capacitance is distributed across the backplane. All the above requirements are

achieved on a six layer, one piece backplane, with a characteristic impedance of the logic tracking complying with that defined in the VMEbus specification.

3 Standards and Approvals

UNICORN is aimed at a worldwide market, so compliance where many national and international standards are considered mandatory. Typical examples are:

- Underwriters Lab UL478
- Canadian Standards Assc. CSA C22.2 No. 220
- Federal Communication Commission (FCC) Regulation Part 15 subpart J class A
- Special Committee of the International Electrotechnical Commission, for Radio Interference (CISPR) Publication 22
- Verband Deutscher Eletrotechniker (VDE) 0871, 0805 and 0806
- International Electrotechnical Commission (IEC) 950
- British Standard 6301
- Local PTT Approvals

Many aspects of these standards are in-built into our own internal standards, and appropriate clauses regarding compliance with relevant standards always form mandatory clauses of purchase specifications of bought-out parts.

Complete machine approvals are carried out by extensive in-house testing and build appraisal, followed by a review of results by the relevant authority or submission to a recognised test house in the relevant country.

INTERFACES

R.N.S.

The International Computers Ltd. ICL2900 Computer Architecture

(compared with the Burroughs B6700/7700)

by

R.W. Doran

Massey University

New Zealand

Abstract:

The ICL2900 hardware architecture is compared with that of the B6700/B7700. The two systems are based on similar principles and are similar in overall design. However, although the ICL2900 postdates the B6700 by seven years, the machines have independent origins and differ in many important details.

1. Introduction

The International Computers Limited new 2900 series (announced 24th October 1974) has been described as "The most advanced and exciting development in the history of computing" (ICL publicity pamphlet [1]). Although this may be arguably true when applied to Britain it is perhaps a slight exaggeration when applied to the whole world. Nevertheless, the arrival of the 2900 is an extremely important event because it is the first totally-new general purpose computer architecture to be announced for many years.

One fundamental principle of the 2900 design philosophy was that all programming would be performed in high level languages [2]. Once this decision is made regarding a computer system the hardware designers may at their liberty modify the design so that it is suited to the languages used in the system and to the compilers which translate the languages. The hardware design does not have to be understood by applications programmers and other users for they are kept at least one level removed from such details. One purpose of this paper is to see what advantage the ICL2900 designers have taken of their new freedom.

The ICL2900 of 1974 is only the second commercial series of computers to be dedicated to high level programming.* The other is the Burroughs series starting with the B5000 of 1962 [3,4]. The B5000 software implementation language ESPOL did allow the programmer to fall back on machine language programming if necessary; however, the successor to the B5000 series eventually called the B6700/B7700 series, does restrict programming entirely to the high level. The other purpose of this paper is to compare the ICL2900 with the B6700 to determine what has happened to computer architecture in the years since the B6700 was announced (as the B6500) in 1967.

---

* There have been other such computers, notably the Hewlett Packard HP3000.

2. Comparison of Computers

In this comparison I will restrict my attention to a specific level of the hardware of the two computers, in particular, the hardware features which must be understood by a systems programmer. Thus software will not be discussed for it lies outside the field of interest. Other aspects such as logical design and even features such as pipelining are, for the most part, below the level at which systems programmers work so these will not be discussed in detail either.

Given the restricted level of discussion, it is reasonable to wonder whether any meaningful comparison can be made at all. A fast Turing machine and, for example, the IBM360 may appear entirely the same to the end user if the Turing machine is disguised in layers of software. Still, one feels that the statement that "The Turing machine architecture is less suited to general purpose computing than the IBM360" is reasonable, even incontravertable.

Scientific, mathematical, and cost effectiveness arguments cannot be used to prove statements about computer architecture. They may be helpful but any conclusion reached at all can be quickly invalidated by a subtle change in software. In fact, decisions based on such arguments often turn out to be very short-sighted; some examples we shall see later.

The parallel with domestic architecture is not too far-fetched here; if cold science was applied to house design, we would all live in boxes or dormatories. As ordinary architecture must take into account the foibles of human occupants, so computer architecture must consider that the computer system will eventually be the home of operating systems, compilers, applications software and all the attendant programmers. The success of a computer design can really only be determined by the respect in which it turns out to be held by those who live with it.

If it is reasonable that computer architecture, as its domestic counterpart, be regarded as partly an art then a new architecture must be subject to criticism, hopefully of a constructive nature. When we make a statement about the merit of an architecture, we are taking the position of a critic because our statements are not demonstrable facts but merely reasoned opinions based on our experience. In what follows I will make some criticisms of the B6700 and ICL2900 but these should not be regarded as disrespectful or insulting - both systems are carefully worked out well-integrated designs and deserving of the highest praise.

A computer's software is a Procrustean test bed for its hardware (which may seem to be the wrong way round - but read on). If the hardware isn't smart enough then the software distorts the machine as seen by the user to make the overall system more extensive. If the hardware is too smart (e.g. by implementing COBOL pictures which are not the same as those of PL/I) then the software must also go out of its way to eliminate unwanted features. The systems designer has to tread a delicate path between these extremes so that the hardware is just suited nicely to its use - or at least so that stretching and amputating are reduced to a minimum.

In what follows there will be some assertions as to the reasoning behind the design of the two computers. These are based partly, for the ICL2900, on their new range documentation [7] and for the B6700/7700 on published papers [9,10]. However some points are not covered in the literature and remarks on the reasoning behind them must be treated as guesswork.

3. Similarities

The B6700 and ICL2900 are both intended for the same range of applications - they are general purpose computers. "General purpose" may once have had the connotation of 'universal' in the sense of automata theory but the term has almost come to mean "suitable for all purposes except real-time and process control." For a specific purpose, such as 'number crunching', a special computer may be far more cost effective than a general purpose machine. However, most main stream manufacturers produce general purpose computers because of the economies of dealing in a small number of different machines and because most installations, even if mainly involved in number crunching, do perform a range of functions with their computers - some calculation, some data processing, some data communications.

a. Intermodule Architecture

At the outermost levels the ICL2900 and B6700 appear to be remarkably similar. The intermodule architecture is practically the same. Both systems employ a distributed organization which allows a range of power to be obtained by expanding the number of modules and which also, if large enough, results in a 'fail soft' system. See figure 1 (note that ICL and Burroughs use different names for the modules - the names are compared in table 1.)

The most well known range of general purpose computers, the IBM360/370, used many different models to obtain a range of power - multiple processors, where available, being a special feature. In the series being considered, Burroughs and ICL only provide two styles of hardware, the B6700, B7700 and the ICL2970, ICL2980 respectively. Each model provides a range from 1x2 (one memory module x (one central processor and one I/O system)) up to the following maxima

	memory modules x processors	total memory size maximum
B6700*	5 x 6	6M bytes
B7700	8 x 8	6M bytes
ICL2970	3 x 4	6M bytes
ICL2980	4 x 4	8M bytes

The advantages to the vendor of dealing in a limited range of models are obvious. It is acceptable to the user because he is unlikely to expand his needs outside the range of one model in the lifetime of the computer; it is advantageous because expansion is comparatively cheap and is not disruptive.

This ability to expand does raise the initial cost of the computer (though this may be offset by economies of scale) for the hardware has to include capabilities to allow for the maximum configuration. For example, a B6700 memory module has to include logic to resolve priority among requests from six sources, even if only two are used (not to mention the cost of sockets, etc.). Many customers on limited budgets are willing to sacrifice expansion capabilities and fail-soft protection (but not speed) if they obtain the advantages of large systems software. Because of this, Burroughs has brought out a limited version of the B6700, the B6748, limited to one central processor, one I/O subsystem and with expansion restricted.

In order to obtain fail-soft capabilities from the multiple modules, each ICL2900 includes a reconfiguration panel which allows modules to be removed without rewiring. A similar feature is available for the B6700 as an optional extra - a 'controlled reconfiguration system'. The B6700 goes one step further with its 'dynamic reconfiguration system' which will automatically reconfigure the computer in event of failure. In both the B6700 and ICL2900 separate I/O subsystems may be linked to provide multiple paths to critical peripheral devices.

The I/O subsystems of fig. 1 were not specified further because there are some differences which are worth noting. The ICL2900 (fig. 2a)

* The B6700 may have up to 64 memory modules but only 5 memory buses (in fact the B6700 treats the processor/memory path as a 5-capacity single bus rather than as 5 separate buses).

Table I.

Terminology - rough equivalents

ICL2900	B6700
Order code processor	Central processor
Storage multiple access control	(part of each memory module)
Disc file controller	Disk pack controller } not as
Sectored file controller	Disk file optimiser } general
Communications link controller	Data communications processor
Storage access control }	I/O multiplexor (not as flexible)
General peripheral controller }	(and peripheral controllers)

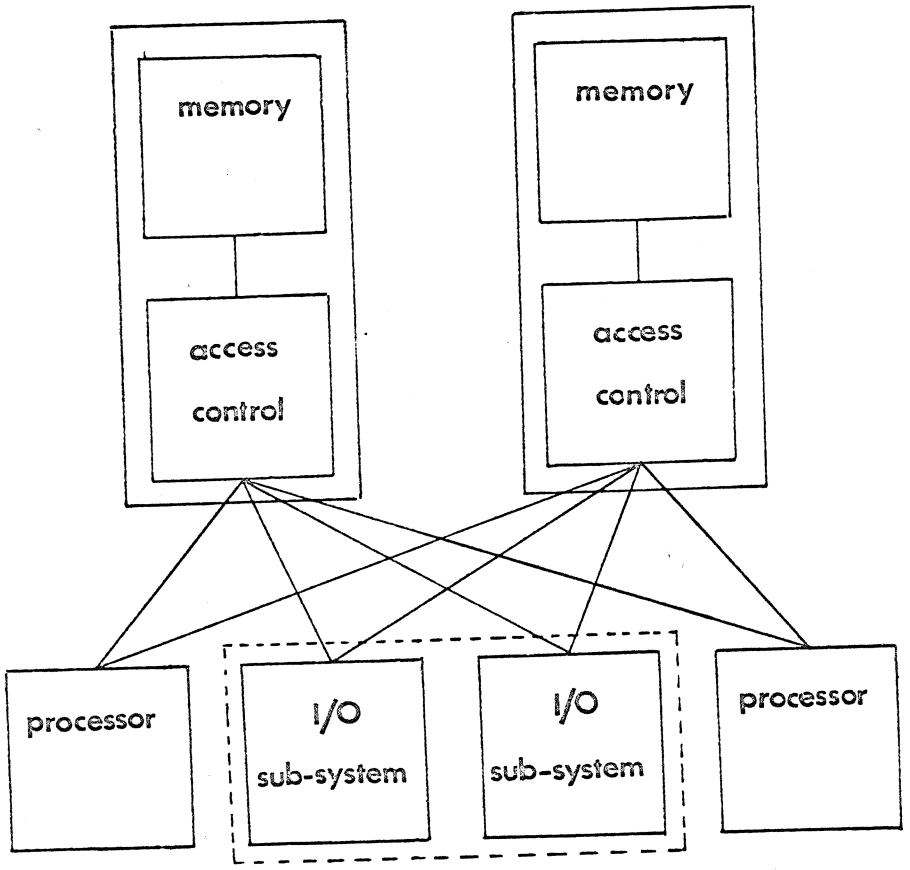


Fig.1 Intermodule architecture

makes greater use of microprogrammed controllers than does the B6700/B7700 (fig. 2b). In the latter systems controllers of devices other than disk packs are hardwired.

In the B6700/B7700, as the controllers in each device are smart enough not to require further intervention, the I/O multiplexor is just that, a conceptually simple device which translates characters, packs them into words and interfaces with memory modules. The ICL2900 breaks the same control functions into different levels. The 'Storage Access Control' interfaces with the memory. Packing and translation, if necessary, are performed in the microprogrammed controllers placed between the SAC peripheral devices. The 'General Peripheral Controller' could be used to take over some of the functions of device controllers, thus resulting in cheaper peripherals.

The B6700 disk file optimiser is in a strange position as it is regarded as a main processor yet actual I/O traffic usually goes through the multiplexor. In the B7700 the I/O module is a programmed device but rather than absorbing the complexity of peripheral controllers these remain the same as the B6700 and the I/O module takes over some functions of the operating system I/O software.

The intermodule architecture of the two computers is soundly designed. That they have essentially the same mechanism may signify the arrival of a new standard. The lesser use made of programmable controllers in the B6700 should be regarded as an indication of its age rather than as a fundamental feature.

b. Within the Central Processors

Here the overall concepts are the same but the details are significantly different. The following general points are in common:

(i) Both systems implement virtual memory and provide a descriptor mechanism to define the virtual memory structure

(ii) Both use hardware stacks for arithmetic, subroutine linkage and execution records

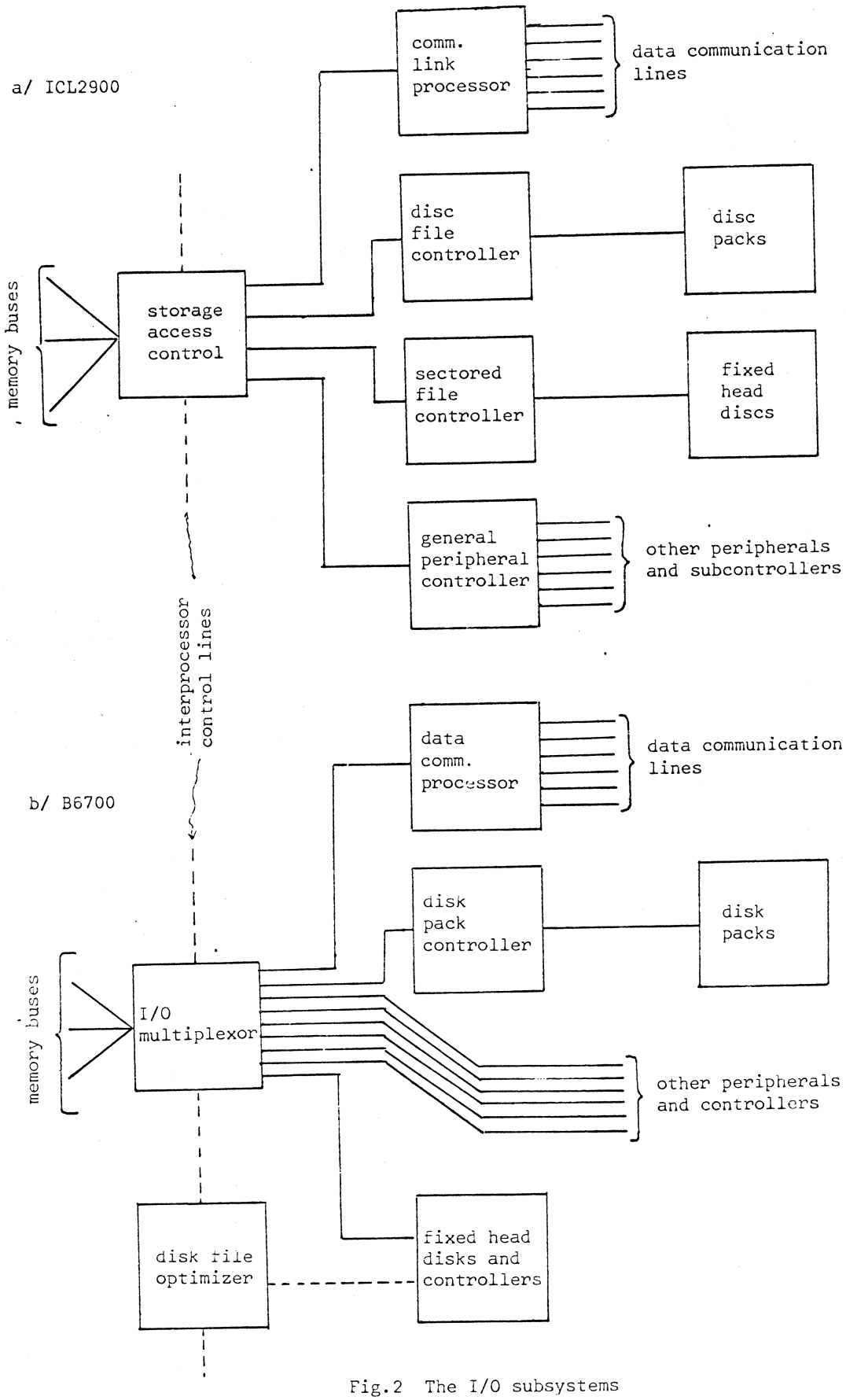


Fig.2 The I/O subsystems

(iii) Both regard the operating system as being part of every task, in which case an interrupt becomes an unexpected subroutine call which is serviced by the interrupted task - an interrupt is not a change of state to a different task. (ICL literature deals with 'virtual machines' and 'processes' rather than tasks.)

(iv) The instruction sets of the basic machines are identical in their extent - the stack instructions are augmented with memory/memory instructions to deal with strings.

4. Differences

This section concentrates on the differences between the two computers, as there isn't space here to investigate the reasons for the many features in common. What follows will therefore be very biased towards pointing out the weaknesses in the architectures. It should be stressed again that both computers have excellent, indeed beautiful, overall designs.

a. Tags

Of the many differences of detail in the two systems by far the most important is that the ICL2900 does not employ a tagged memory like that of the B6700.

In his monograph on the Basic Language Machine, J.K. Iliffe [5] stresses the need for tagging parts of a computer's memory (a tag being a few bits associated with each word of memory which give its type). Iliffe uses tags to distinguish absolute addresses which must be updated when the location of virtual memory in absolute memory is altered (whether within main storage or between main and backing storage). He does point out other advantages:

i. that individual words can be protected in a segment of words of mixed type

ii. instructions can take advantage of their knowledge of the type of operands they are dealing with.

The 2900's virtual memory mechanism eliminates the need to search for absolute addresses and this makes the need for tags not so obvious. The lack of protection mentioned in point i. is serious but it is not fatal because of the ICL protection mechanism discussed below. The lack of operand dependence, point ii. above, has one immediate effect on the architecture - compared with the B6700 the ICL2900 has relatively simple instructions.

Why an important feature such as tags should be omitted from a new computer is hard to understand, but I suspect that it is a hardware

decision made without proper regard to software problems. Tags are not very often necessary as the types of most words are obtainable from the descriptors of the segments to which they belong. What is very obvious is the cost of tags - an extra, say, four bits attached to every word of 32 bits could increase the cost of memory by up to 10%. In fact, fewer instructions are needed when tags are employed, because each instruction can do more. This results in smaller programs and a decrease in memory requirement - but this is very hard to measure at the design stage.

The B6700 has a larger word size (48 bits) so the waste of space for tags is not so critical. It could be eliminated entirely by using tags only in sequences, such as stacks, which contain items of mixed types. This would require that logical word boundaries be different from the physical boundaries in the main storage. This is a complication but not necessarily a cause of inefficiency in fast machines where the memory is accessed in large chunks and via slave or cache stores.

In this day and age lack of tags is a sad deficiency. Although the lack of protection is partially side stepped it does have some unfortunate side effects on the ICL2900 design.

b. The Arithmetic Stack

In its use of a stack for arithmetic the 2900 diverges slightly from the B6700 in that it follows the example of the MU5 [6]. Rather than employing a hidden mechanism to reduce transfers between the memory and top of stack registers, the MU5 and 2900 provide accumulators which must be explicitly referenced by the programmer or the compiler to produce an optimal sequence of transfers (see fig. 3). As the B6700 has tags, single and double word transfers between the stack and the special registers are performed automatically - the distinction between single, double and quadruple words in the case of the ICL2900 has to be maintained by the programs.

The invisible mechanism of the B6700 is certainly a great boon to the compiler writers. It is difficult to imagine how such a scheme

could be implemented without tags and automatic conversion between types, though the 2900 accumulator is heading in this direction for it can hold data of different types and be set to three different lengths. Instructions can take the accumulator length into account but there must be different instructions for each type.

The ICL2900 instructions are not usually cast completely in Polish postfix form, e.g. to form a+b on top of the stack one would normally use:-

"push accumulator into stack and load accumulator with a"
"add b to the accumulator"

rather than in the B6700:-

"push a into the stack"
"push b into the stack"
"add"

This has the apparent advantage that fewer instructions are needed but the less apparent disadvantage that the operation codes of instructions which refer to memory must be larger (because there are more of them).

It should be stressed that the single visible accumulator of the ICL2900 offers no efficiency advantages. An expression may be optimized by converting it to an equivalent form which, when translated directly to reverse Polish form,requires less stack space, e.g. a←b*(c+e*f) to (e*f+c)*b → a. However, this optimization will have the same effect whether the top of stack accumulator is visible or hidden as in the B6700. The B6700 mechanism has the advantage that it will have the effect of automatically reducing stack/memory references - this is particularly true in the case of the B7700 with its 32 top of stack registers. The top of stack registers really act as a special slave store. The accumulator of the ICL2900 can be regarded similarly, though it is the only slave store to be visible at the architectural level (the current models have slave stores for the whole stack).

The machines also vary in their approach to decimal arithmetic - the ICL2900 implements it but the B6700 doesn't! The main problem with

decimal arithmetic is that decimal numbers are of varying lengths whereas stacks are of fixed width. The 2900 allows decimal arithmetic on operands of 8, 16 or 32 packed digits. Both machines include scaling and conversion operators but the B6700 requires that all arithmetic be between binary operands. The B6700 method, although low level, isn't too silly because the time taken to obtain unpacked digits and pack them is similar to that taken to convert them to binary form (i.e. memory access time is the bottleneck).

One of the 2900s 'top of stack' registers may be used as an index register. This is lacking in the B6700 and is a serious deficiency which tends to make calculations involving arrays comparatively slow. To overcome this, as an optional extra, the B6700 may be provided with 'vector mode' instructions which, as with string processing instructions, by-pass the stack for address calculations. Vector mode still has some deficiencies such as only being able to deal with three vectors at one time. Of course, cache memories in the B7700 and 2900s make ordinary memory locations 'fast' and thus overcome the lack of high speed index registers.

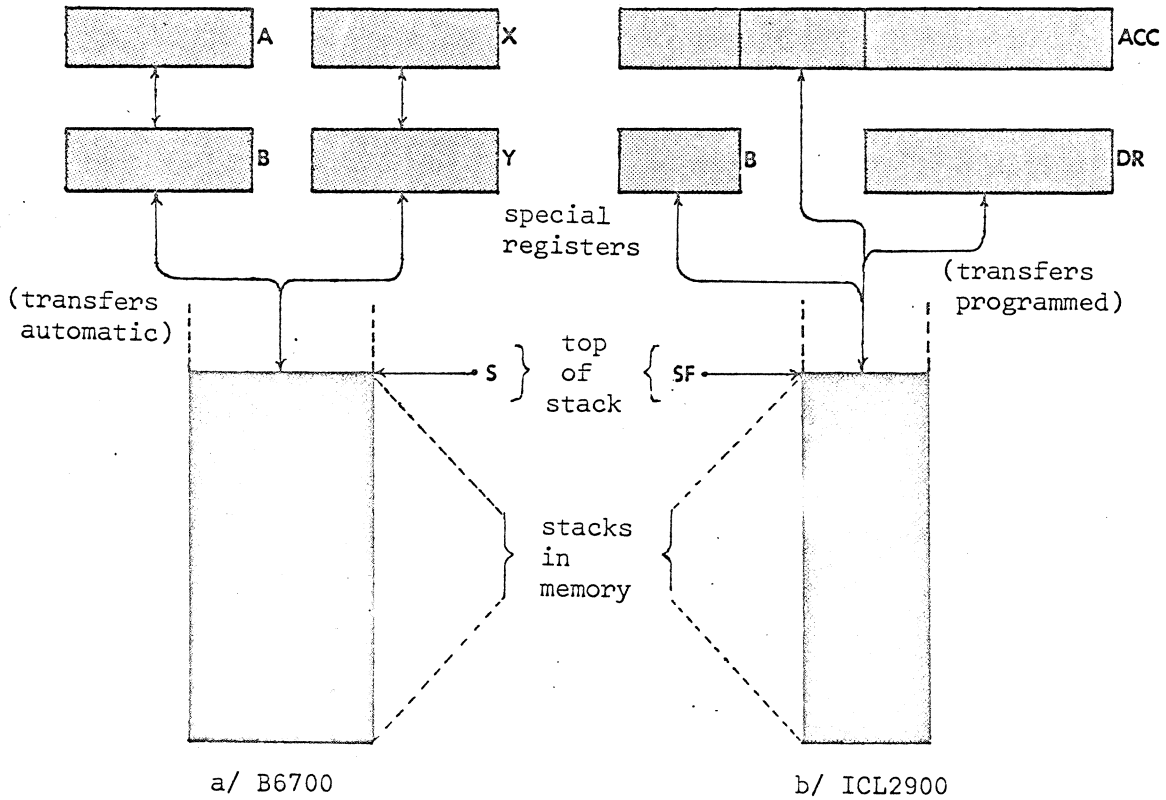


fig.3 Arithmetic stack mechanisms

In summary, the B6700 stack mechanism is cleaner than that of the ICL2900. Most differences, however, are due to the original decision in the 2900 to do without tags.

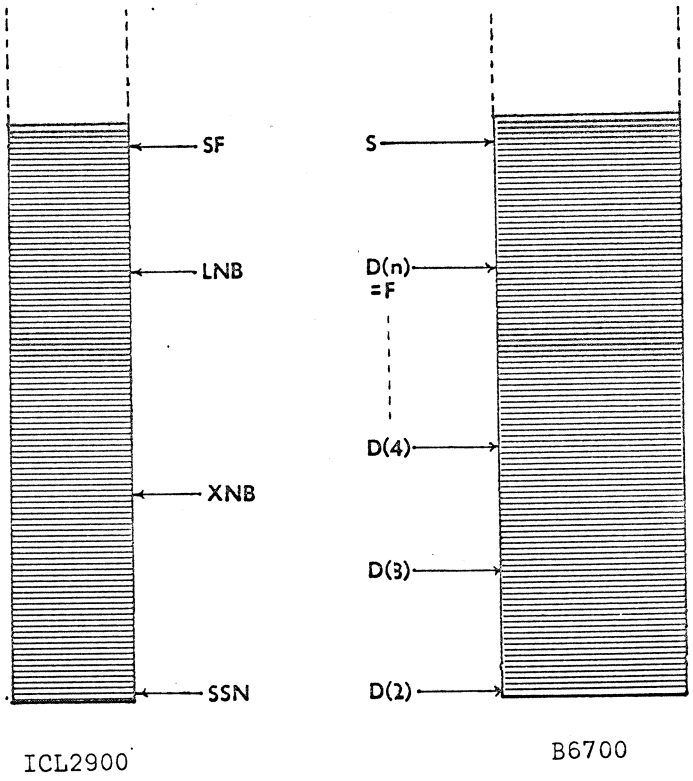


Fig.4 Addressing within stacks.

c. Stacks as Execution Records

Both machines make use of their arithmetic stacks for subroutine linkage and temporary data storage. Instruction sequences are provided for entering subroutines and for returning from them. Linked words are created and placed in the stack thereby defining the subroutine stack within the arithmetic stack. The B6700 sequence is somewhat simpler because the individual instructions are more extensive.

For addressing data within the stack the ICL2900 provides three registers but leaves the maintenance of the block structure display up to software - apart from LNB being set to point to local variables and SSN to globals, both automatically (see fig. 4). The B6700, on the other hand, imbeds display maintenance in the hardware, providing an infinity (i.e. 30) of display registers for addressing within the stack.

The ICL decision not to provide display registers seems to be based partly on the observed fact that most programs do not make use of any but global and local variables with perhaps some on one other level; it seems hard to justify the cost of including unused registers. However, that is a very hardware-oriented point of view - what is overlooked is that software must maintain the display even if it is only used occasionally - for the ICL2900 this means that subroutine entry and exit protocols must be much more complicated than would otherwise be the case.

While it is true that the applications languages most widely used - Fortran and COBOL - are not highly structured, both the B6700 and ICL2900 have chosen to use highly-structured languages for systems programming-versions of Algol 60 and Algol 68 respectively. With these languages the need for displays is more obvious - it is rumoured, for example, that the B6700 Algol compiler uses seven levels of block structure at its deepest point.

The main function of display registers is to provide bases for relative addressing. It is usual, however, to regard them as also being faster than main storage. The ICL2900 does not lose out in this regard because even though display registers are held in main memory (in the stack) they are frequently used and will most often be obtained from fast slave stores.

Another argument against implementing displays is that the concept is not well enough understood yet and it may be an unnecessary rococo feature which will be discarded in future. This isn't very likely, though it could well be that display registers will be extended to other uses. Furthermore, display registers are well suited to the

current algorithmic languages and all current languages may be assisted a great deal by a common hardware mechanism.

While on the subject of execution records it is worth noting that both machines have a feature whereby an operand access may be turned automatically into a function call. The B6700 included this in order to implement the Algol 'call by name' parameter mechanism whereas the ICL2900 regards it as an automatic escape, often to the operating system, for making the hardware seem to be at a higher level than it really is.

d. Virtual Memory

The two machines provide similar descriptor mechanisms which allow a process to create its own tree-structured address space. A descriptor defines each area of virtual memory and gives various facts about the elements of the area, for example, the types of the elements. It is heartening to find descriptors becoming accepted although the scheme of the B6700 and ICL2900 has some deficiencies (e.g. it is often useful to have a descriptor based relative to a more extensive segment).

Both machines implement paging but with different attitudes. The B6700 mainly uses variable sized unpaged segments but allows data segments to be paged into 256-word chunks. With the 2900, paging of segments is normal but variable sized unpaged segments are allowed (albeit in 128 byte increments). See fig. 5 which illustrates the different styles of virtual memory.

The two computers diverge in that the B6700 maps the virtual memory structure directly into the real memory whereas the ICL2900 uses an intermediate large linear virtual memory. The most important effects of this difference are that -

i. the ICL2900 requires two memory references to convert every virtual address to a real address and needs an associative memory to make this at all viable. (The less often used paging of the B6700 also takes extra memory references - with the B7700 the slave memories would eliminate these.)

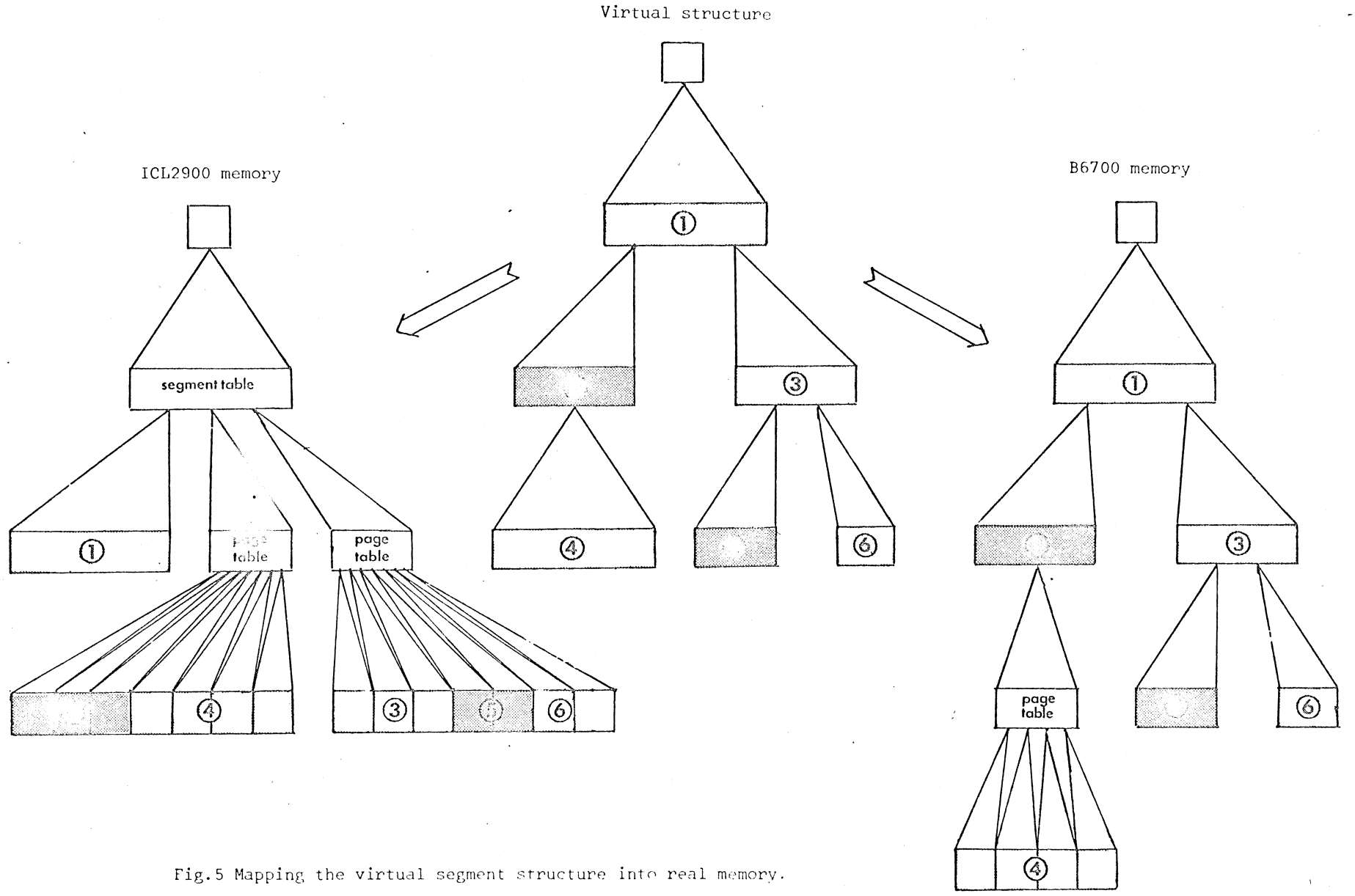


Fig.5 Mapping the virtual segment structure into real memory.

ii. the B6700 operating system must search its stacks for addresses to be updated whenever a virtual segment is moved in real memory (special instructions are provided to assist with this).

The 2900 is equally flexible but a lot cleaner than the B6700 - the efficiency of the B6700 was bought at the cost of software complexity. The 2900 is not perfect, however, for some of the attributes which should be specified in descriptors, such as levels of protection (see below), are in fact associated with segments of the real memory.

e. Protection

The descriptor mechanism used in both computers provides a natural basis for protection. References into a segment are always checked to ensure that they are in range. Furthermore, type information in each descriptor may be used to restrict access, e.g. to stop programs being treated as data. Of course, if a program can alter descriptors then the protection is lost. This is a serious problem because, unless storage is to be static, it must be possible for a program to in fact create and modify descriptors.

The B6700 provides some further protection by the use of a special tag for descriptors - no user program in the normal course of execution can alter a descriptor. However, there are instructions which enable this to be performed. Compilers are regarded as an extension of the operating system and are trusted to generate sensitive instructions in safe circumstances only. Protection in the B6700 depends on the correct operation of all system software. A bug can easily cause the whole system to crash and, therefore, experimental compilers, etc. cannot be tested during normal operation. Although absolute protection is not guaranteed, the vast majority of compiler and system bugs are caught by the combined protection of descriptors and tags, and by programming being restricted to high-level languages using trusted compilers.

The ICL2900, having no tags, cannot guarantee ever to protect a task from itself. Any sensitive information within a stack may be over-

written if a compiler generates bad code. As well as descriptors, items such as return addresses could be inadvertently destroyed thereby causing a program to execute chaotically. This is one of the unfortunate results of 'taglessness'.

In the 2900, although a task can destroy its own stack, the whole system is protected by a multiple level protection scheme. Each segment of memory is assigned a level of protection from 0 to 15 and each task a level of trustworthiness in the same range. The hardware checks the level of a task against that of segments it tries to access and restricts the set to which it may write and from which it may read. As virtual addresses in the stack of a task in error may be accidently altered, these protection levels have to be associated, most unnaturally, with the segments of virtual memory rather than those defined by descriptors. (The virtual memory tables may only be altered by the privileged kernel programs at the innermost levels of protection.)

Further ramifications of stacks being unprotected arise in the treatment of subroutines. If a procedure calls one of a less trusted level, then the new procedure must be given a new stack of its own for it must not be allowed to commit patricide by destroying items in the stack of its parent.

Although they are better than most other machines, the protection mechanisms of both computers leave something to be desired. The B6700 requires too much software to be correct. The 2900 doesn't help a program catch its own errors. Clearly, it would be possible to design a system which has the advantages, but not the disadvantages, of both the B6700 and the ICL2900.

f. Interrupts

Both machines, as mentioned before, handle interrupts as subroutine calls executed in the stack of the task which the processor just happens to be executing. As they relegate fielding of real-time interrupts to peripheral processors, neither machine has a real need for a priority interrupt mechanism where there are a hierarchy of

interrupt routines, however, five levels are still provided by the ICL2900.

The B6700 treats all of its many interrupts alike but the ICL290 further divides its into two classes. Interrupts which naturally belong to a task are handled in that task's stack but other interrupts cause an automatic transfer to an alter-ego stack, one of which is maintained for each task. This is a hard feature to explain. A surface reason is that 'stackfull' interrupts have to be serviced somewhere but the B6700 manages to do this by allocating a few words extra at the end of each stack segment. The extra ICL2900 stacks do not save any space compared with the B6700 though the B7700 does by having one special overflow stack for each processor.

The real reason for duplicate stacks may be the lack of protection for particular words in a stack. With two stacks, sensitive information about each task may be kept protected in the second stack - in the B6700 this may just as safely be kept in the one stack. The strange stack feature may then be ascribed as an indirect effect of the decision not to include tags.

Neither machine has a feature which allows interrupts such as underflow to be redirected by the hardware to routines of the task itself. Thus the ICL virtual machines are not completely independent but must use the same operating system kernel. Incidentally, the B6700 does not allow underflow interrupts to be masked out - an annoying deficiency, especially for numerical processing. (Vector mode doesn't generate underflow interrupts at all.)

g. Intertask communication

The use of speed-up mechanisms isn't of much concern at this level but it is interesting how some problems are handled. If two tasks are communicating by a variable which happens to be held currently in a local store of each processor, then how does one get to know that the other has altered its copy?

The most general method of handling this would be for all local memories to communicate with each other - a solution which currently is not very practical. As it has no cache storage the B6700 doesn't suffer from this problem. The B7700 gets around it by causing all processors to disgorge their local memories back into the main memory whenever one processor executes the synchronising 'read with lock' instruction. Thus any reference to an updateable shared variable should be preceded by a 'read with lock' instruction.

The 2900 uses a similar but smarter mechanism. There are a pair of synchronising instructions but what is significantly different is that all addresses in local memories may be flagged to indicate that they should be thrown out when either one of the two instructions is executed.

The instructions for synchronization are still primitive but are perhaps adequate at the present stage of development of operating system concepts. One would expect future computers to provide a whole set of synchronizing instructions rather than the few implemented at present.

5. Conclusions

By implementing a similar architecture ICL has confirmed that many of the contraversial aspects of the Burroughs B6700 are soundly based and may become the accepted norm. However, the older B6700 is a higher level machine than the ICL2900 mainly because of the 2900's lack of tagged memory. It is disappointing that no real advances have been made in architecture in the seven years between the 6500 and the 2900. However, this is to be expected as the ICL2900 appears to be derived mainly from the MU5 and Iliffe's work, both of which are of 1968 vintage (with a MULTICS influence which is even earlier).

It has been claimed that the ICL2900 merely follows the B6700 [8], but from what we have seen this does not bear close examination. Apart from showing that it could be done, the B6700 has had little direct effect on the 2900; the direct influences mentioned above are insularly British. It is therefore especially heartening that there has been a considerable convergence of ideas.

Acknowledgement

Thanks is due to International Computers (New Zealand) Ltd., for providing access to documentation and to John Deas and John Gibson of ICL for discussions on the 2900.

References

1 ICL Publicity Brochure
"The Way Ahead"

2 ICL Publicity Pamphlet
"Technical overview of 2900 series"

3 Lonergan, W. & King, P.
"Design of the B5000 System"
Datamation Vol. 7 No. 5, pp. 28-32, May 1961

4 Burroughs B6700 Reference Manual
Form 1058633, Burroughs Corp. Detroit 1972

5 Iliffe, J.K.
"Basic Machine Principles"
American Elsevier 1968

6 Kilburn, T., Morris, D., Rohl, J.S., Sumner, F.H.
"A System Design Proposal"
Proc IFIPs Cong. 1968, pp. 806-811

7 ICL New Range System Documentation
In particular Vol. 2.3

8 Dorn, P.
"ICLs new brand of 'Me-Too-Ism'"
Datamation, Dec. 1974

9 Barton, R.S.
"Ideas for Computer Systems Organization - a Personal Survey"
in "Software Engineering", Vol 1 Academic Press 1970

10 Creech, B.A.
"Architecture of the B6500"
Proc. 1969. COINS Symposium, pp. 29-44

DESCRIPTORS

Type 0 Vector

	0 0	Size	0	USC	BCI	Bound
m.s.	0——1	2——4	5	6	7	8 31
l.s.	0	Byte Address				31

Size 0 = 1 bit
3 = 8 bits
5 = 32 bits
6 = 64 bits
7 = 128 bits

USC = unscaled If USC = 0 when a modifier is added to address field it (the modifier) is scaled according to size field:-
2, 3, 4 places UP for 32, 64, 128 respectively
3 places DOWN for 1 bit.

BCI = Bound check inhibit if =1. If BCI=0 Bound field should be 1 greater than largest modifier.

Type 2 Descriptor as for type 0 except bits 0, 1 = 10 and only size 64 allowed.

Type 1 String

	0 1	0 1 1 0 0 0	Length
m.s.	0 7	2——7	8——31
l.s.	0	Byte Address	31

Length = length of string, in bytes, whose 1st byte is given by contents of address field.

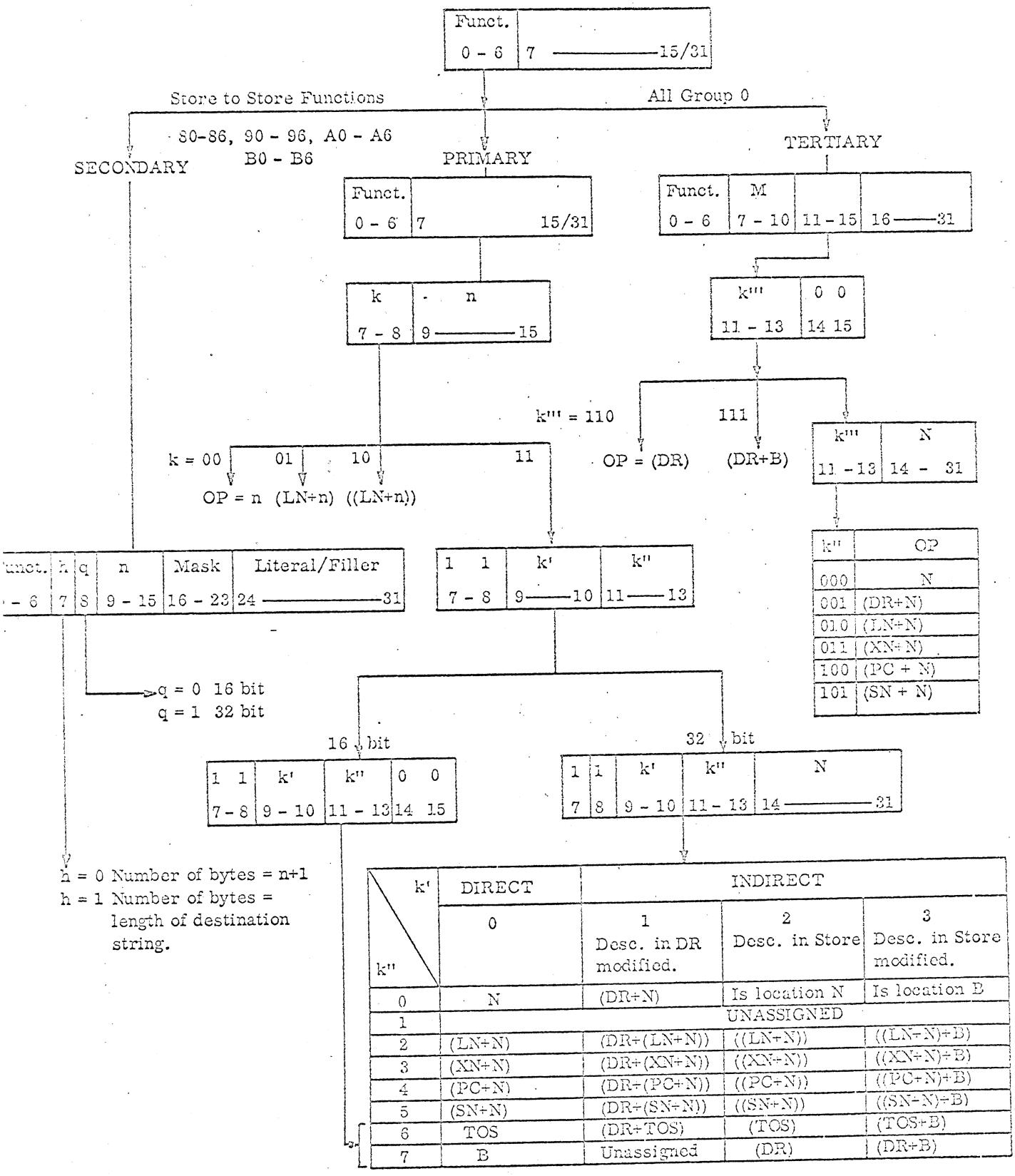
Type 3 Miscellaneous Bits 2-7 define a subtype number

Subtype 32	bounded code
33	unbounded code
35	system call
37	Escape

NOTATION

XX'	=	New contents of XX after instruction has been obeyed.
(XX)'	=	New contents of location specified by XX after instruction has been obeyed.
((XX))'	=	New contents of location addressed by descriptor specified by XX.
TOS	=	Top of stack item.
Lit.	=	n = a 7 bit signed literal.
	or	N = an 18 bit signed literal.
OV	=	Overflow register.
CC	=	condition code register.

DESCRIPTOR FORMAT (TO R



k''	OP
000	N
001	(DR+N)
010	(LN+N)
011	(XN+N)
100	(PC + N)
101	(SN + N)

k' / k''	DIRECT	INDIRECT		
	0	1 Desc. in DR modified.	2 Desc. in Store	3 Desc. in Store modified.
0	N	(DR+N)	Is location N	Is location B
1	UNASSIGNED			
2	(LN+N)	(DR+(LN+N))	((LN+N))	((LN+N)+B)
3	(XN+N)	(DR+(XN+N))	((XN+N))	((XN+N)+B)
4	(PC+N)	(DR+(PC+N))	((PC+N))	((PC+N)+B)
5	(SN+N)	(DR+(SN+N))	((SN+N))	((SN+N)+B)
6	TOS	(DR+TOS)	(TOS)	(TOS+B)
7	B	Unassigned	(DR)	(DR+B)

RSD 2.5.1. ISSUE 2/0)

FIG. 2.

Grp. Func.	0	1	2	3	4	5	6	7
0	Illegal T	VAL 10 Validate Address Set CC P 8.1.5.9	ADB 20 B'=B+OP OV P 8.1.4.4		SL 40 TOS = ACC ACC's = OP $\overline{OV}$ P 8.1.3.5	SLD 50 TOS = DR DR' = OP CC' = (DR)' type P 8.1.5.2	L 60 ACC's = OP $\overline{OV}$ P 8.1.3.6	LDRL 70 DR' = OP + OP addr. CC' = (DR)' type P 8.1.5.4
2	JCC 02 Jump if M_{cc} = 1 T 8.1.2.13	CYD 12 ACS' = 64 ACC' = DR $\overline{OV}$ P 8.1.3.10	SBB 22 B' = B-OP OV P 8.1.4.4	MPSR 32 Modify PSR P 8.1.3.1	SLSS 42 TOS = ACC ACS' = 32 ACC' = OP $\overline{OV}$ P 8.1.3.4	SLB 52 TOS = B B' = OP $\overline{OV}$ P 8.1.4.2	LSS 62 ACS' = 32 ACC' = OP $\overline{OV}$ P 8.1.3.3	LDA 72 DR' = OP 32-63 P 8.1.5.5
4	JAT 04 Jump if M cond. is true T 8.1.2.14	INCA 14 DR'=DR+OP 32-63 32-63 P 8.1.5.10	DEBJ 24 B' = B - 1 Jump if B' ≠ 0 OV P 8.1.2.12	CPSR 34 Copy PSR to OP P 8.1.3.2	SLSD 44 TOS = ACC ACS' = 64 ACC' = OP $\overline{OV}$ P 8.1.3.4	TDEC 54 OP' = OP-1 Set CC P 8.1.2.7	LSD 64 ACS' = 64 ACC' = OP $\overline{OV}$ P 8.1.3.3	LDTB 74 DR' = OP 0-31 P 8.1.5.6
6	JAF 06 Jump if M cond. is false T 8.1.2.14	MODD 16 Modify Descriptor P 8.1.5.8	CPB 26 B=OP; CC = 0 B<OP; CC' = 1 B>OP; CC' = 2 P 8.1.4.5		SLSQ 46 TOS = ACC ACS' = 128 ACC' = OP $\overline{OV}$ P 8.1.3.4	INCT 56 OP' = OP + 1 Set CC P 8.1.2.7	LSQ 66 ACS' = 128 ACC' = OP $\overline{OV}$ P 8.1.3.3	LDB 76 DR' = OP 8-31 P 8.1.5.7
8	T	DIAG 18 Diagnose p 9.2.1	SIG 28 If CC=0 OP' = a new descriptor. Set CC = 1 P 8.1.5.11	EXIT 38 Procedure Exit OP = 7 bit lit. P 8.1.2.9	ST 48 OP' = ACC P 8.1.3.7	STD 58 OP' = DR P 8.1.5.3	RRTC 68 ACS = 64 ACC' = value of RT clock. $\overline{OV}$ P 8.1.3.11	LD 78 DR' = OP CC' = (DR)' type P 8.1.5.1
A	T	J 1A Jump PC' = PC + lit. v OP_{0-31} P 8.1.2.11	MYB 2A B' = B. OP OV P 8.1.4.4	ESEX 3A Escape Exit P 8.1.2.15	STUH 4A OP' = m.s. ½ of ACC. ACS' = ½ ACS P 8.1.3.9	STB 5A OP' = B P 8.1.4.3	LUH 6A ACS' = 2. ACS m.s. ½ of ACC' = OP. $\overline{OV}$ P 8.1.3.8	LB 7A B' = OP $\overline{OV}$ P 8.1.4.1
C	T	JLK 1C TOS = PC and Jump. P 8.1.2.10	VMY 2C B'=(OP-x)y x, y, z=(DR) $\overline{OV}$ P 8.1.4.7	OUT 3C Causes a Class 9 Interrupt. P 8.1.2.16		STLN 5C OP' = SSN/LNB P 8.1.2.5	RALN 6C LNB' = SF - OP. P 8.1.2.3	LLN 7C LNB' = OP P 8.1.2.1
E	T	CALL 1E Procedure Call P 8.1.2.8	CPIB 2E B-OP;CC' = 0 B<OP;CC' = 1 B>OP;CC' = 2 B' = B + 1 OV P 8.1.4.6	ACT 3E Activate P 9.2.2	IDLE 4E Await Any Interrupt P 8.1.2.17	STSF 5E OP' = SSN/SF P 8.1.2.6	ASF 6E SF' = SF + OP P 8.1.2.4	LXN 7E XNB' = OP P 8.1.2.3

NOTES P - Primary Orders
S - Secondary Orders
T - Tertiary Orders
Function numbers are in hexadecimal.
References are to NRSD 2.5.1.

OV or $\overline{OV}$ refers to state of overflow after the order.
INT = Fixed point format.
FP = Floating point format (Real)
DEC= Decimal format.
LOG= Logical format.

8	9	A	B	C Logical	D Decimal	E Integer	F Real
TCH 80 Table Check Set CC S 8.3.3.7	PK 90 Pack OV S 8.3.3.9	SWEQ A0 Scan while Equal Set CC S 8.3.3.1	MVL B0 Move Literal S 8.3.3.6	UAD C0 ACC' = ACC + OP. Set CC LOG $\overline{OV}$ P 8.2.5.1	DAD D0 ACC' = ACC + OP. DEC OV P 8.2.6.1	IAD E0 ACC' = ACC + OP. INT OV P 8.2.4.1	RAD F0 ACC' = ACC + OP. FP OV P 8.2.3.1
ANDS 82 And Strings S 8.3.3.12	INS 92 Conditional Insert S 8.3.3.11	SWNE A2 Scan while Unequal Set CC S 8.3.3.2	MV B2 Move Strings S 8.3.3.4	USB C2 ACC' = ACC - OP. Set CC LOG $\overline{OV}$ S 8.2.5.2	DSB D2 ACC' = ACC - OP. DEC OV P 8.2.6.1	ISB E2 ACC' = ACC - OP. INT OV P 8.2.4.1	RSB F2 ACC' = ACC - OP. FP OV P 8.2.3.1
ORS 84 Or Strings S 8.3.3.12	SUPK 94 Suppress and Unpack. Set CC $\overline{OV}$ S 8.3.3.10	CPS A4 Compare String. Set CC S 8.3.3.3	CHOV B4 Check String Overlap. Set CC S 8.3.3.5	URSB C4 ACC' = OP - ACC. Set CC LOG $\overline{OV}$ P 8.2.5.3	DRSB D4 ACC' = OP - ACC. DEC OV P 8.2.6.2	IRSB E4 ACC' = OP - ACC. INT OV P 8.2.4.2	RRSB F4 ACC' = OP - ACC. FP OV P 8.2.3.2
NEQS 86 Not Equivalence Strings S 8.3.3.12	EXP 96 Expand S A.2.2.1	TTR A6 Table Translate S 8.3.3.8	COM B6 Compress S A.2.2.1	UCP C6 Logical Compare. Set CC P 8.2.5.4	DCP D6 Decimal Compare. Set CC P 8.2.6.3	ICP E6 Fixed Point Compare. Set CC P 8.2.4.3	ACP F6 Floating Point Compare. Set CC P 8.2.3.3
EXPA 88 Expand ACC P A.2.2.2	COMA 98 Compress ACC P A.2.2.2	FLT A8 Combine ACC with OP' to form FP number. OV P 8.2.4.11	FIX B8 Convert to Integer Set CC $\overline{OV}$ P 8.2.3.10	USH C8 Shift ACC OP. places. LOG $\overline{OV}$ P 8.2.5.5	DSH D8 Shift ACC 4. OP places. DEC OV P 8.2.6.4	ISH E8 Shift ACC OP. places. Set CC INT OV P 8.2.4.4	RSC F8 Scale OV P 8.2.3.4
AND 8A ACC' = ACC & OP $\overline{OV}$ P 8.2.5.6	DDV 9A ACC' = $\frac{ACC}{OP}$ DEC $\overline{OV}$ P 8.2.6.6	IDV AA ACC' = $\frac{ACC}{OP}$ INT OV P 8.2.4.6	RDV BA ACC' = $\frac{ACC}{OP}$ FP OV P 8.2.3.6	ROT CA Rotate ACC OP. places. $\overline{OV}$ P 8.2.5.7	DMY DA ACC' = ACC. OP. DEC OV P 8.2.6.5	IMY EA ACC' = ACC. OP. INT OV P 8.2.4.5	RMY FA ACC' = ACC. OP. FP OV P 8.2.3.5
OR 8C ACC' = ACC v OP. $\overline{OV}$ P 8.2.5.6	DRDV 9C ACC' = $\frac{OP}{ACC}$ DEC $\overline{OV}$ P 8.2.6.7	IRDV AC ACC' = $\frac{OP}{ACC}$ INT OV P 8.2.4.7	RRDV BC ACC' = $\frac{OP}{ACC}$ FP OV P 8.2.3.7	SHS CC Shift l.s. 32 bits ACC OP. places $\overline{OV}$ P 8.2.5.8	DMYD DC ACC' = ACC. OP. ACS' = 2. ACS $\overline{OV}$ P 8.2.6.9	IMYD EC ACC' = ACC. OP. ACS' = 2.ACS Set CC $\overline{OV}$ P 8.2.4.9	RMYD FC ACC' = ACC. OP. ACS' = 2.ACS OV P 8.2.3.9
NEQ 8E ACC' = ACC ≠OP $\overline{OV}$ P 8.2.5.6	DMDV 9E ACC' = $\frac{ACC}{OP}$ TOS=remainder Set CC DEC $\overline{OV}$ P 8.2.6.8	IMDV AE ACC' = $\frac{ACC}{OP}$ TOS=remainder Set CC INT OV P 8.2.4.8	RDVD BE ACC' = $\frac{ACC}{OP}$ ACS' = $\frac{1}{2}$ ACS FP OV P 8.2.3.8	SHZ CE Shift left till m.s. bit ACC≠0. OP=No. $\overline{OV}$ P 8.2.5.9	CBIN DE Convert ACC to binary OV P 8.6.2.10	CDEC EE Convert ACC to decimal ACS' = 2.ACS $\overline{OV}$ P 8.2.4.10	Illegal P

SUMMARY (TO NRSD 2.5.1 AT 4/0)

FIG.1

JUMPS

J	8.1.2.11	1A	P	Jump unconditionally. PC' = PC + LIT or OP_{0-30}
DEBJ	8.1.2.12	24	P	Decrement B & jump if non-zero. B' = B-1, jump if B'≠0.
JCC	8.1.2.13	02	T	Jump on CC. PC' = PC + LIT or OP_{0-30} if bit (CC) of M = 1

M =	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
JCC can be	-	3	2	2 3	1	1 3	1 2	1 2 3	0	0 3	0 2	0 2 3	0 1	0 1 3	0 1 2	A N Y

JAT	8.1.2.14	04	T	Jump on arithmetic true. jump if 'M' is true.
JAF	8.1.2.14	06	T	Jump on arithmetic false. jump if 'M' is false.

Where with floating M = 0 ACC = 0, M = 1 ACC > 0, M = 2 ACC < 0
with fixed M = 4 ACC = 0, M = 5 ACC > 0, M = 6 ACC < 0
with decimal M = 8 ACC = 0, M = 9 ACC > 0, M = 10 ACC < 0
with B M = 12, B = 0, M = 13, B > 0, M = 14, B < 0
and M = 11, DR BOUND = 0, M = 15, OV = 1

JLK	8.1.2.10	1C		Jump & link. TOS' = PC, PC' = PC + LIT or OP_{0-30}
CALL	8.1.2.8	1E	P	Procedure call. (LN + 1)' = descriptor of PC and jump
EXIT	8.1.2.9	38	P	Procedure exit. stack returned to status quo, jump to link
ESEX	8.1.2.15	3A	P	Escape exit. PC' = TOS, D' = 1 with descriptor in DR

DR INSTRUCTIONS

LD	8.1.5.1	78	P	Load DR. DR' = OP_{0-63}, CC' = descriptor type
LDRL	8.1.5.4	70	P	Load relative. DR' = OP_{0-63} + OP address, CC' = descriptor type
LDA	8.1.5.5	72	P	Load address. $DR'_{32-63} = OP_{0-31}$
LDTB	8.1.5.6	74	P	Load type and bound. $DR'_{0-31} = OP_{0-31}$
LDB	8.1.5.7	76	P	Load bound. $DR'_{8-31} = OP_{8-31}$
SLD	8.1.5.2	50	P	Stack & load DR. TOS' = DR, DR' = OP_{0-63}, CC' = descriptor type
STD	8.1.5.3	58	P	Store DR. OP'_{0-63} = DR
INCA	8.1.5.10	14	P	Increment address. $DR'_{32-63} = DR_{32-63} + OP_{0-31}$
MODD	8.1.5.8	16	P	Modify DR. $DR'_{8-31} = DR_{8-31} - OP_{8-31}$, $DR'_{32-63} = DR_{32-63} + OP_{0-31}$ scaled.
VAL	8.1.5.9	10	P	Validate address. check validity of DR by OP_{8-11} as ACR OK CC' = 0, | read only CC' = 1, | write only CC' = 2, | invalid CC' = 3.
SIG	8.1.5.11	28	P	Start significance. If CC' = 0, OP_{0-63} = type 1 descriptor of length = 1. address = address - 1 and CC set to 1. If CC ≠ 0, do nothing.

B INSTRUCTIONS

LB	8.1.4.1	7A	P	Load B. B' = OP_{0-31}, OV' = 0
SLB	8.1.4.2	52	P	Stack & Load B. TOS' = B, B' = OP_{0-31}, OV' = 0
STB	8.1.4.3	5A	P	Store B. OP'_{0-31} = B, OV' = 0
ADB	8.1.4.4	20	P	Add to B. B' = B + OP_{0-31}, OV' = overflow
SBB	8.1.4.4	22	P	Subtract from B. B' = B - OP_{0-31}, OV' = overflow
MYB	8.1.4.4	2A	P	Multiply B. B' = B. OP_{0-31}, OV' = overflow
CPB	8.1.4.5	26	P	Compare B. B = OP_{0-31}, CC' = 0, | B < OP, CC' = 1, | B > OP, CC' = 2 |
CPIB	8.1.4.6	2E	P	Compare B increment B. as CPB & B' = B + 1
VMY	8.1.4.7	2C	P	Dope vector multiply. B' = (OP_{0-31} - x). y, check $0 \leqslant B' < z$ & $0 < (OP - x) < 2^{31}$, x, y, z = (DR), (DR + 1), (DR + 2)

STACK CONTROL

LLN	8.1.2.1	7C	P	Load LNB. LNB' = OP_{14-29}, OP_{0-13} must equal SSN
STLN	8.1.2.5	5C	P	Store LNB. OP'_{0-31} = SSN/LNB
RALN	8.1.2.3	6C	P	Raise LNB. LNB' = SF - OP_{0-31}
ASF	8.1.2.4	6E	P	Adjust SF. SF' = SF + OP_{0-31}, OP may be -ve
LXN	8.1.2.2	7E	P	Load XNB. XNB' = OP_{0-29}
STSF	8.1.2.6	5E	P	Store SF. OP'_{0-31} = SSN/SF

SUNDRIES

RRTC	8.1.3.11	68	P	Read real time clock. OP'_{0-63} = clock output.
OUT	8.1.12.16	3C	P	Out. Cause class 9 interrupt with OP_{0-31} as parameter
IDLE	8.1.12.17	4E	P	Idle. Suspend instruction sequencing until interrupt occurs
ACT	9.2.2	3E	P	Activate. OP_{0-63} = LSTB; OP_{64-95} is spare; OP_{96-110} = SSN. Load CPU registers with contents of SSN + 1 and start process.

STRING INS

SWEQ	8.3.3.1	A0	S	Scan while equal. (DR) > ref. CC' = 2
SWNE	8.3.3.2	A2	S	Scan while unequal.
CPS	8.3.3.3	A4	S	Compare strings. CC' = 2 | (DR) < (A
MV	8.3.3.4	B2	S	Move. (DR) string =
CHOV	8.3.3.5	B4	S	Check overlap. non (ACC) < (DR), CC' =
MVL	8.3.3.6	B0	S	Move literal. (DR)'
TCH	8.3.3.7	80	S	Table check. check if bit = 1 setting CC'
TTR	8.3.3.8	A6	S	Table translate. (D
PK	8.3.3.9	90	S	Pack. ACC' = (DR)
SUPK	8.3.3.10	94	S	Suppress & unpack.
INS	8.3.3.11	92	S	Conditional insrt.
ANDS	8.3.3.12	82	S	And string. (DR)' st
ORS	8.3.3.12	84	S	Or string. (DR)' stg
NEQS	8.3.3.12	86	S	Not equivalent strin literal.

SEMAPH

INCT	8.1.2.7	56	P	Increment and test
TDEC	8.1.2.7	54	P	Test and decrement OP = 0, CC = 0 | O OP = - 1, CC = 3 | (OP = final value for

AC

L	8.1.3.6	60	P	Load. ACC' = OP, AC
LSS	8.1.3.3	62	P	Set ACS 32 & load.
LSD	8.1.3.3	64	P	Set ACS 64 & load.
LSQ	8.1.3.3	66	P	Set ACS 128 & load.
LUH	8.1.3.8	6A	P	Load upper half. AC
CYD	8.1.3.10	12	P	Copy DR. ACS' = 64
SL	8.1.3.5	40	P	Stack & load. TOS'
SLSS	8.1.3.4	42	P	Stack, set ACS 32 & ACC = OP_{0-31}, OV'
SLSD	8.1.3.4	44	P	Stack, set ACS 64. ACC = OP_{0-63}, OV'
SLSQ	8.1.3.4	46	P	Stack, set ACS 128 & ACC = OP_{0-127}, OV
ST	8.1.3.7	48	P	Store. OP_{ACS} = ACC
STUH	8.1.3.9	4A	P	Store upper half. OP OV' = 0
MPSR	8.1.3.1	32	P	Modify PSB. If OP_{2} CC' = OP_{28-29} | I or OP_{25} = 1, PM' =
CPSR	8.1.3.2	34	P	Copy PSR. OP_{0-15} OP_{28-29} = CC' | O

L

UAD	8.2.5.1	C0	P	Logical add. ACC' =
USB	8.2.5.2	C2	P	Logic subtract. AC
URSB	8.2.5.3	C4	P	Logical reverse sub borrow, OV' = 0
UCP	8.2.5.4	C6	P	Logical compare. ACC > OP, CC = 2
USH	8.2.5.5	C8	P	Logical shift. Shift OP-ve, OV' = 0
ROT	8.2.5.7	CA	P	Rotate. Rotate AC C OV' = 0
SHS	8.2.5.8	CC	P	Shift 32 bits. Shift OP-ve, OV' = 0
SHZ	8.2.5.9	CE	P	Shift while zero. Sh places, OV' = 0
AND	8.2.5.6	8A	P	And. ACC' = ACC &
OR	8.2.5.6	8C	P	Or. ACC' = ACC v
NEQ	8.2.5.6	8E	P	Not equivalent. AC

TRUCTIONS

T ≠ (DR) stg, not found CC' = 0
(DR) < ref. CC' = 3
IT = (DR) stg, not found CC' = 0 else 1
R) = (ACC), CC' = 0 | (DR) > (ACC),
C), CC = 3 |
(ACC) string
CC' = 0 | (ACC) > (DR), CC' = 1 |
2 |
tring = literal
DR) stgs with bit in (ACC) stg and stop
= 1, else CC' = 0
) string is translated by (ACC) stg.
string, packed. OV = 0
(DR)' string = ACC unpacked, CC is set
R)' string = LIT (CC = 0) or mask (CC ≠ 0)
g. = (SR) stg. & (ACC) stg. or literal
= (DR) stg. v (ACC) stg. or literal
. (DR)' stg = (DR) stg ≠ (ACC) stg. or

DRE INSTRUCTIONS

P' = OP + 1
OP' = OP - 1
> 0, CC = 1 | OP < - 1, CC = 2 |

INCT and original value for TDEC).
INSTRUCTIONS

, OV' = 0
ACS' = 32, ACC' = OP_{0-31}, OV' = 0
CS = 64, ACC' = OP_{0-63}, OV' = 0
ACS = 128, ACC' = OP_{0-127}, OV' = 0
' = 2ACS, ms ½ of ACC' = OP_{ACS}, OV' = 0
ACC' = DR, OV' = 0
= ACC, ACC' = OP_{ACS}, OV' = 0
Load. TOS_{ACS} = ACC, ACS' = 32,
= 0
Load. TOS_{ACS} = ACC, ACS' = 64,
= 0
Load. TOS_{ACS} = ACC, ACS' = 128,
= 0
, OV' = 0
$_{ACS}$ = ms ½ of ACC, ACS' = ½ ACS.

= 1, ACS' = OP_{30-31} | If OP_{26} = 1,
OP_{24} = 1, PM' = 1's at OP_{16-23} = 1's
's at OP_{16-23} = 0's.
= 0 | OP_{16-23} = PM' | OP_{24-27} = 1110 |
$_{30-31}$ = ACS'.

GICAL (WORD)

ACC + OP_{0-31}, CC' = carry, OV' = 0
' = ACC - OP_{0-31}, CC' = borrow, OV' = 0
ract. ACC' = OP_{0-31} - ACC, CC' =

CC = OP_{ACS}, CC' = 0 | ACC < OP, CC' = 1 |

ACC left OP_{25-31} places or right if

left OP_{25-31} places or right if OP-ve,

ACC_{0-31} left OP_{25-31} places or right if

ift ACC left until ACC, ≠ 0, OP' = no. of

OP_{ACS}, OV' = 0
OP_{ACS}, OV' = 0
' = ACC ≠ OP_{ACS}, OV' = 0

DECIMAL

DAD	8.2.6.1	D0	P	Decimal add. ACC' = ACC + OP_{ACS}, OV' = overflow
DSB	8.2.6.1	D2	P	Decimal subtract. ACC' = ACC - OP_{ACS}, OV' = overflow
DRSB	8.2.6.2	D4	P	Decimal reverse subtract. ACC' = OP_{ACS} - ACC, OV' = overflow
DMY	8.2.6.5	DA	P	Decimal multiply. ACC' = ACC.OP_{ACS}, OV' = overflow
DMYD	8.2.6.9	DC	P	Decimal multiply double. ACS' = 2ACS, ACC' = ACC.OP_{ACS}, OV' = 0
DDV	8.2.6.6	9A	P	Decimal divide. ACC' = ACC/OP_{ACS}, OV' = 0
DMDV	8.2.6.8	9E	P	Decimal remainder divide. ACC' = ACC/OP_{ACS}, TOS_{ACS} = rem. OV' = 0, rem = 0 or rem > 0 & div > 0, CC' = 0,| rem > 0 & div < 0, CC' = 1,| rem < 0 & div > 0, CC' = 2,| rem < 0 & div < 0, CC' = 3 |
DRDV	8.2.6.7	9C	P	Decimal reverse divide. Acc' = OP_{ACS}/ACC, OV' = 0
DCP	8.2.6.3	D6	P	Decimal compare. ACC = OP_{ACS}, CC' = 0 | ACC < OP, CC = 1 | ACC > OP, CC = 2 |
DSH	8.2.6.4	D8	P	Decimal shift. Shift ACC left 40P places or right if -ve, OV' = overflow
CBIN	8.2.6.10	DE	P	Convert to binary. ACC' (binary) = ACC (pk decimal), OV' = overflow
CDEC	8.2.4.10	EE	P	Convert to decimal. ACS' = 2 ACS, ACC' (dec) = ACC (bin), OV' = 0

FIXED POINT (INTEGER)

IAD	8.2.4.1	E0	P	Add. ACC' = ACC + OP_{ACS}, OV' = overflow
ISB	8.2.4.1	E2	P	Subtract. ACC' = ACC-OP_{ACS}, OV' = overflow
IRSB	8.2.4.2	E4	P	Reverse subtract. ACC' = OP_{ACS} - ACC, OV' = overflow
IMY	8.2.4.5	EA	P	Multiply. ACC' = ACC.OP_{ACS}, OV' = overflow
IMYD	8.2.4.9	EC	P	Multiply double. ACS' = 2ACS, ACC' = ACC.OP_{ACS}, OV' = 0 ACC + ve, OP + ve, CC' = 0,| ACC + ve, OP - ve, CC' = 1,| ACC - ve, OP + ve, CC' = 2 | both - ve, CC = 3 |
IDV	8.2.4.6	AA	P	Divide. ACC' = ACC/OP_{ACS}, OV' = overflow
IRDV	8.2.4.7	AC	P	Reverse divide. ACC' = OP_{ACS}/ACC, OV' = overflow
IMDV	8.2.4.8	AE	P	Remainder divide. ACC' = ACC/OP_{ACS}, TOS' = rem, 'CC' as DMDV
ICP	8.2.4.3	E6	P	Compare. ACC = OP_{ACS}, CC' = 0,| ACC < OP, CC' = 1,| ACC > OP, CC' = 2 |
ISH	8.2.4.4	E8	P	Arithmetic shift. ACC' = ACC.2^i, i = OP_{26-31}. OV' = overflow i + ve, CC' = 3 | i - ve, all last bits = 0, CC' = 0 | Last bit 0, CC' = 1 | Last bit 1, CC' = 2 |

FLOATING POINT (REAL)

RAD	8.2.3.1	F0	P	Floating add. ACC' = ACC + OP_{ACS}, OV' = overflow
RSB	8.2.3.1	F2	P	Floating subtract. ACC' = ACC - OP_{ACS}, OV' = overflow
RRSB	8.2.3.2	F4	P	Floating reverse subtract. ACC' = OP_{ACS} - ACC, OV' = overflow
RMY	8.2.3.5	FA	P	Floating multiply. ACC' = ACC.OP_{ACS}, OV' = overflow
RMYD	8.2.3.9	FC	P	Floating multiply double. ACS' = 2ACS, ACC = ACC.OP_{ACS}, OV' = overflow
RDV	8.2.3.6	BA	P	Floating divide. ACC' = ACC/OP_{ACS}, OV' = overflow
RRDV	8.2.3.7	BC	P	Floating reverse divide. ACC' = OP_{ACS}/ACC, OV' = overflow
RDVD	8.2.3.8	BE	P	Floating divide double. ACS' = 2ACS, ACC' = ACC/OP_{ACS}, OV' = overflow
RCP	8.2.3.3	F6	P	Floating compare. ACC = OP_{ACS}, CC' = 0 | ACC < OP, CC' = 1| ACC > OP, CC = 2 |
RSC	8.2.3.4	F8	P	Scale. ACC' = ACC.16^i . i = OP_{22-31}, OV' = overflow
FLT	8.2.4.11	A8	P	Float. ACS' = 2 ACS, ACC'_{FP} = ACC int. i = OP_{22-31} (biased) OV' = overflow
FIX	8.2.3.10	B8	P	Fix. ACS' = ½ ACS, ACC' int = ACC_{FP}, OP' = ACC exp (unbiased) OV' = 0

EMULATION

COMA	A2.2.2	98	P	Compress ACC ACC' (6 bit chars) = ACC (8 bit bytes)
EXPA	A2.2.2	88	P	Expand ACC ACC' (8 bit bytes) = ACC (6 bit chars)
COM	A2.2.1	B6	S	Compress (ACC)' stg. (6 bit) = (DR) stg. (8 bit)
EXP	A2.2.1	96	S	Expand (DR)' stg. (8 bit) = (ACC) stg. (6 bit)

(TO NRSD 2.5.1 ISSUE 4/0)

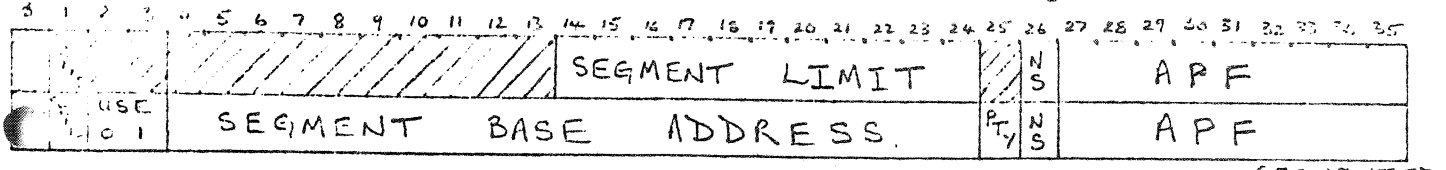
FIG.3

2.1.8 TRANSLATION UNIT GENERAL DESCRIPTION

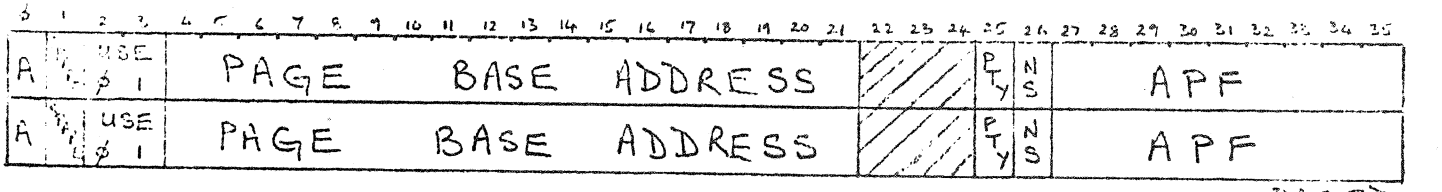
(Refer to 2.5.0,O.D.-P9Ø)

Translation requests originate from either the Operand Slave (OS) or the Stack Slave (SS) or the Instruction Slave (IS). Simultaneous accesses are dealt with in the order OS first then SS and then IS. The virtual form of the address is loaded at access time into VF. Association between VF (Ø-13) [if access is being made to an unpaged segment] or between VF (Ø-2Ø) [if access is being made to a paged segment] and the contents of the CAMs selects the appropriate double word (36 bits) RAM entry.

These RAM lines hold information as illustrated in figure 1.



SEGMENTED



PAGED

FIG 1.

To obtain the real address bits VF (14-27) are added to the segment base address or bits VF (22-27) are concatenated with the page base address.

The other RAM bits are used as follows:

1) A:- A=Ø means the page is not available in core.

2) USE bits:- USEØ=1 indicates segment or page has been referenced.
USE1=1 " " " " " " written to.

3) NS:- NS=Ø segment may be freely slaved.

4) APF:- Access protection field. Access to a segment is permitted if
READ and APF (5-8) ≥ ACR.
WRITE and APF (1-4) ≥ ACR.
EXECUTE and APF (Ø) = 1.

5) SEGMENT LIMIT:- VF (14-24) ≥ segment limit indicates a segment limit violation.

The translator stops processing a particular access if there is a violation detected of cases 1, 4, 5, as described above. A NEQ marker is set (1, 5) and this is passed back to the slave originating the translation request. From there it is pipelined down to the modifier where a virtual store interrupt is initiated (VSI).

If there is a usage change (see 2 above) the translator updates its own RAM copy of the appropriate USE bits and the NEQ marker is set along with a UUB marker. These are again passed to the originating slave and pipelined to the modifier where a dummy table search is initiated which enables the appropriate core store table entry use bits to be updated.

An APF failure (see 4 above) causes a fail marker to be set and this is sent to the originating slave (OS or IS. SS is always assumed accessible) from where it is pipelined down to the modifier to cause a program error interrupt PEI.

If no association is found between VF (Ø-13) in the segmented case, VF (Ø-2Ø) in the paged case and the CAMs the NEQ marker is set and when this is returned to the originating slave and pipelined down to the modifier it initiates the table search sequence. This references the table entries in core store by switching the OS to perform real address mode read accesses via the translator. These accesses by-pass the slave portion of the translator and are passed straight from VF by the appropriate gating waveforms to SR. Strobing SR starts off the request to store.

SEQUENCING

To overlap accesses in the translator a sequencing arrangement is provided. This stores the type of access (read or write), which slave made the request (IS, SS, OS) and whether data available should be inhibited from being sent to the slaves (IHO, IHS), and a marker to show which accesses have not been fully dealt with (VAL) in a series of 4 bit // shift registers. By this means 1 access may be held in the SMAC buffer, a second hold in SR a third in VW, VT and a fourth in VF all at their various levels of being processed.

WRITES

On access the virtual address is loaded into VF. If a previous write is still waiting for its D/AV the virtual address stays in VF. If VF, VW are not equivalent then the previous write data when available is cleared to BW the address from VT (VT holds the translated form write address) to SR, VF is then transferred to VW where VW* reselects the entry gating to SE, SO for the 2nd write.

VF VW=

If a read access is made to the translator and the information in VW is valid and VF, VW are equivalent then all further accesses are held up until the Write data has arrived and the read data has been sent from SMAC.

If a write access is made to the translator and VF, VW are equivalent then AW is overwritten on those bytes which have their write byte markers set. No store request is made.

Translator Slave Size

The slave portion of the translator consists of 2 sets of CAMs, 8 lines deep and each line associating on 21 bits. Each line points to a double line RAM entry. The sets can be used individually or combined to give 16 lines. Rejection of any line is done on the basis of least use. If 16 lines are used rejection occurs between the 2 sets alternately.

OPERAND SLAVE ACCESSES

The OS as well as requesting read or write access as already described also makes various other types of access.

1) Check:- The check access simply causes an APF check on the WRITE permission of the segment to which the OS address belongs.

2) Clear:- A clear access causes the translator to unload any outstanding valid writes. Since writes are cleared to store by the next write to a different four word block there will usually be 1 outstanding uncleared write just before a change of process. The clear access ensures that this write is performed.

Various combinations of Read Write Check and Clear also occur as follows:

A) Read and Clear.

B) Read and Check.

C) Check and Clear.

D) Check and Read and Clear.

E) Write and Clear.

3) Check zero:- This access causes a zero check on the data on SE, SO, to be carried out. The check may be selected to be on SE only or SE and SO.

4) Dummy Write:- For certain operations controlled by the modifier the only available piepline routing is via SE, SO to the operand buffers and back into the pipeline. For those cases a dummy write type access is made by the OS which causes the translator to select the appropriate gating to SE, SO, as in the case of a normal write, but no write to store occurs.

5) PC Access:- For accesses from the OS where the operand is specified as a PC+N type the OS makes a PC access. This makes the translator do an execute protection check on the OS address to check that the operand referenced is in fact from a code segment.

The hardware/software interface of the ICL 2900 range of computers

D. H. R. Huxtable and J. M. M. Pinkerton

International Computers Limited, Lovelace Road, Bracknell, Berkshire RG12 4SN

The paper aims to outline those ICL 2900 hardware features that support system software. The treatment broadly is in terms of the instruction set and other hardware features. 2900 architecture provides for concurrent execution of independent processes in virtual machines, so that the processes are protected from mutual interference. The protection arrangements are outlined together with ways of addressing the store, for which several kinds of descriptors may be used. There are both visible and invisible hardware registers; access to the latter is privileged via a so called image store addressing mechanism. The arrangements for handling interrupts and entering procedures are outlined. Slave stores are provided for reducing store access demands for frequently accessed items. Peripheral transfers are via autonomous and asynchronous controllers activated by processes operating within a central processor. In conclusion some of the main considerations influencing design decisions are reviewed.

(Received April 1977)

1. Introduction

The ICL 2900 range was designed with both the hardware and the operating system software in mind; that is, with a view to presenting an overall system to the user. This paper contains an account of the underlying architecture with special reference to those features of the instruction set that are designed to support the software. It is presented as a hardware description, but it is really a description of an interface and the implementation in actual hardware of the features described may vary from one model in the range to another: for example some registers may in one model be composed of bistables and in another be simply locations in main memory that are used by the hardware for the purpose.

At the primitive level the hardware supports:

1. A virtual memory with segmentation and paging.
2. A process stack.
3. Memory protection with a series of levels of increasing privilege and with special features to permit the running of diagnostic programs during normal operations.
4. A comprehensive interrupt and fault handling system closely integrated with the procedure call mechanism.
5. Slave (buffer) stores made safe in relation to channel transfers and multiprocessor operation by a consistent system of hardware provision and software convention.

The above hardware interface is designed to support an operating system that is stack oriented and provides each user with a virtual machine of his own. The writer of subsystems on which the ultimate high level user depends has available to him, insofar as he requires them, similar hardware support features to those that are available to the writer of the operating system itself.

2. The process stack and storage access

Virtual addresses contain 32 bits of which 14 define the number of the segment. The remaining bits are divided into page number, word number, and byte number. In order to address whole words only the first thirty bits are necessary.

A number of the registers listed in Appendix 1 serve the purpose of supporting a stack of which there is one associated with each process. The *stack segment number* register (SSN) is loaded by the operating system—specifically by the ACTIVATE instruction—with the segment number of the stack when a process is activated. SSN contains 14 bits and when concatenated with 16 zeros gives the virtual address of the base of the stack. The first 14 bits of the *stack front register* (SF) and the *local name base register* (LNB) are identical with those of SSN and these registers therefore contain pointers into the stack segment. SF, as its name implies, points to the stack front (more exactly to the next available location) while LNB always points to somewhere within the stack so that LNB < SF. The convention adopted in this paper is that the abbreviation may stand either for the register or its content. Conventionally LNB is used by the operating system to point to the working area of the current procedure; when a new procedure is called the old value of LNB is recorded at the top of the stack and LNB updated so as to point to the working area of the new procedure. Instructions that manipulate the stack by placing items on it, by removing items from it, or by making an explicit adjustment to SF carry with them hardware checks that lead to an interrupt if the condition LNB < SF is violated.

The instruction set permits direct access to be made to items on the stack by specifying their addresses relative to LNB. Only locations at or above that pointed to on LNB can be addressed in this way and the hardware checks that they are below SF.

In addition to LNB there are two other registers, the *extra name base register* (XNB) and the *cross-reference table base register* (CTB). These are of the full 30 bits and while they may be used to point in to the stack they are not generally so used. XNB may sometimes be used for keeping track of a former name space when a new one has been created by a procedure entry. Direct access to the store may be made relative to XNB or CTB in exactly the same way as reference is made relative to LNB. Since these references are not necessarily to the stack there can be no hardware checking at this point. However, the checking facilities provided by the virtual storage accessing mechanism for ensuring that accesses do not go outside segment limits are available.

The hardware is also capable of making access to the store via a *vector descriptor*. Vector descriptors consist of two words and specify a consecutive set of items in the store. The items must all be of the same size which can be one bit, one byte, one word, two words, or four words. Three bits in the first word of the descriptor give the size and other bits give the length of the vector, that is, the number of items that it contains. The second word contains the virtual address of the base of the vector. The *accumulator* (ACC) is four words long, but it may be set to operate if desired at the shorter lengths of two words or one word. The combined effect of these facilities is that strong

support is given for single, double and quadruple length working.

Access to an item via a vector descriptor may be *direct* or *indirect*. For the former kind of access to be possible there must be a copy of the descriptor in the *descriptor register* (DR), it having either been placed there by an explicit instruction or left there as the result of a previous operation. The vector may then be accessed using as an index (offset) either a literal in the instruction or the content of a storage location specified relative to one of the following: LNB, XNB, CTB, *program counter* (PC). As an alternative, an *indirect access* may be made via a descriptor contained in the store. The location containing the descriptor is specified relative to one of the registers just mentioned. The first stage of the operation is to transfer this descriptor to DR; a second access is then performed, the offset, if any, being given by the number in the *Index Accumulator* (B).

A modifier used to give an offset to the base address given in a descriptor is first checked against the length field and an interrupt caused to occur if the check fails. The modifier is then scaled according to the size given in the size field and added to the base address. Checking and scaling will, however, not take place if inhibit bits in the first word of the descriptor are set.

More detail about the various ways in which the store may be addressed will be found in Appendix 2. In addition to vector descriptors there are string descriptors, descriptor descriptors, code descriptors, system call descriptors, and escape descriptors. These are all distinguishable by certain combinations of bits in the first word.

3. Access to the image store

It is a feature of the design that all hardware registers should be addressable by suitably privileged programs. To this end the hardware registers are conceptually grouped together to form what is known as the *image store*, in which each register has its own address. Some registers are only accessible in this way and are known as *invisible* registers. Others take part in operations called for by nonprivileged instructions available to the ordinary programmer. These are known as *visible* registers, the most obvious example being the accumulator. The functions of the most important registers in the image store will be explained below. Appendix 1 contains a reference list of registers in the image store, together with the abbreviations that stand for them.

The image store gives the appearance of being a consecutively addressed set of 32-bit locations. Image store locations may be addressed as such by using appropriate instructions from the regular instruction set, with image store addresses formed in the way described in Appendix 2. Such instructions count as privileged instructions. Some image store locations are by their nature read-only; for example, that corresponding to the real time clock. Locations corresponding to the visible registers are read-only, since these registers can be written into by using unprivileged machine instructions.

Access to the image store is controlled by two bistables known as PRIV and ISR. If PRIV = 1 (that is if the bistable is set) both read and write access—where the latter is possible—is allowed. When PRIV = 0 no writing access is permitted, subject to an exception concerning diagnostic programs that will be explained later. If ISR = 1 read-only access to the image store is permitted.

PRIV is itself represented by a bit in one of the image registers namely the *program status register* (PSR); other bits in this register serve such purposes as indicating whether arithmetic overflow has occurred, what the current size of the accumulator is, and so on.

ISR is represented by a bit in the *system status register* (SSR); other bits in this register identify the type of processor in use (this information is wired in and therefore unalterable), mask interrupts, control a real addressing mode that permits the segment tables to be bypassed when an initial load is being performed, and serve other similar purposes.

It will be seen that PRIV and SSR, being themselves bits in invisible registers, can be changed by program only when PRIV = 1. PRIV becomes set equal to 1 automatically when an interrupt occurs or when an explicit system call is made, the latter being equivalent to an interrupt. The system routine entered after an interrupt thus enjoys complete privilege. Having serviced the interrupt it will make use of an ACTIVATE instruction to restart the interrupted process or perhaps to restart some other previously suspended process. One of the functions of the ACTIVATE instruction is to set PSR and SSR so as to give the activated process its correct degree of privilege. Thus ACTIVATE can reduce privilege but not increase it; the same applies to an EXIT instruction used to return from a procedure.

It is desirable in the operation of a large computer system to be able to test peripheral devices and the circuits associated with them while the system is running a normal load. This is done by inserting a diagnostic program which runs as a job under the system. So that it should be able to perform adequate checks, this program must be able to write into buffer registers and other invisible registers associated with the device that it is testing. With the system described above it would be necessary for the diagnostic program to be given complete privilege and this is clearly undesirable. Accordingly it is arranged that access to registers needed for diagnostic purposes can be obtained if another bit known as DGW in SSR is set, provided that a diagnostic allow bit associated with the register or registers concerned is also set. This system enables diagnostic programs to be run with a minimum of danger to system integrity.

4. Virtual store addressing and protection

Implementation of the virtual memory makes use of segment tables and page tables in a manner that is now familiar. The base of the segment table corresponding to the currently operating process is held in an invisible register known as the *local segment table base register* (LSTB). There is a second invisible register, the *public segment table base register* (PSTB). Local segments—local, that is, to a process, although they may be shared with other processes—are those in the range 0 to 8191 and public segments are those in the range 8192 to 16383. Thus, whenever a segment is being accessed, the hardware refers to LSTB or PSTB according to the value of the most significant digit in the segment number. In 2900 terminology a local segment table is said to define a *virtual machine*; a number of virtual machines may share the same public segment table. Within a virtual machine each process has its own stack.

Each segment table entry consists of two words which, in addition to the effective address of the segment and the segment size, contain information about whether the segment is paged or not paged and whether it is shared or not shared. If the segment is shared the effective address points to the relevant entry in a *global segment table*. If the segment is not shared, the effective address points either to the base of the segment if it is not paged or to the base of the relevant page table if it is paged. In order to provide information to the store management system the hardware sets bits in the page tables of paged segments to indicate whether pages have been read from or written into. In the case of unpaged and unshared segments similar bits are set in the segment table.

A nine bit field in the segment table entry is known as the *access protection field* (APF) and is used for virtual store protection. Four of these bits constitute the *read access lock* (t_1), four constitute the *write access lock* (t_2), and one is the execute permission bit (t_3). When access to a segment is attempted a check is made that $t_1 >$ ACR or $t_2 >$ ACR as the case may be, ACR being a register containing the access key for

the domain of a process in execution. Failure of this check leads to an interrupt. A segment may not be executed as code unless $t_3 = 1$.

Exceptions to the above rules are made in the case of read or write accesses made by a process to its own stack. A local segment may thus be given very low values of t_1 and t_2 and thus be protected from possible corruption by procedures running at higher levels, while yet remaining accessible to a procedure, running at similar levels, that uses it for its stack. This does not lead to any loss of protection, since the stack segment is protected by the way in which it is generated and also by certain checks that take place on stack register operations. The stack is, in fact, largely manipulated by hardware mechanisms.

5. The interrupt mechanism

Interrupts in the 2900 range are classified as asynchronous or synchronous. Asynchronous interrupts come from outside, typically from a mechanical peripheral or from a clock. Synchronous interrupts, which in other systems are often referred to as traps, occur during the operation of an instruction. They can arise either from some unexpected happening such as an accumulator overflow or a program error, or they can be deliberately provoked as, for example, in a CALL instruction, when it is necessary for the operating system to be entered at a high level of privilege to adjudicate on some matter of protection. Thus, the 2900 range handles all program changes that require the attention of the operating system in a uniform manner whether they arise as a result of an interrupt or are explicitly called for by the programmer.

Synchronous interrupts take effect as soon as they occur, a bit in SSR being set if the current instruction is incomplete. Asynchronous interrupts are dealt with according to an order of priority depending on the class to which they belong. Various internal interrupts, of which accumulator overflow is typical, may be masked by setting bits in PSR. System interrupts, including virtual store interrupts, may be masked by setting bits in SSR.

The *event pending* bit (EP) in SSR enables the system programmer to arrange that a delayed interrupt shall occur, for example, on emergence from a system procedure. If the EP bit is set, such an interrupt occurs on the execution of any EXIT instruction that causes ACR to be increased. The interrupt is masked however, if the appropriate bit in SSR is set. An interrupt can also occur in certain circumstances if the EP bit is set when a process is reactivated by an ACTIVATE instruction.

On receiving an interrupt, the hardware brings about a forced entry into a procedure designed to service the interrupt, first storing the process and program status on the stack so that it can be restored on return. Some interrupts can be processed using the same stack as the interrupted program; calls to the operating system are included under this heading. Others, such as external interrupts and virtual store interrupts, require that a temporary switch to another stack should take place. Interrupts are divided into classes and for each class there is an entry in a table known as the *interrupt steering table* (IST), there being one such table for each processor in a multiprocessor system. The method of saving program and system status depends on whether the stack has to be switched or not. The IST has entries for each interrupt class, giving the values that must be loaded into the relevant hardware registers for the interrupt processing to proceed. Some details of the various classes of interrupts are given in Appendix 3.

6. Procedure entry

There are three ways of entering a procedure. To enter a procedure running in the same name space and in the same protection environment a simple jump and link instruction may be used; no interrupt is involved. A similar result may be produced without an explicit jump instruction through the operation of the escape mechanism. This occurs when, during the course of the execution of an instruction, an escape descriptor is found to be in, or comes to enter, DR. Execution of the instruction is then cancelled and the computer proceeds to execute the code pointed to by the escape descriptor. It is possible for the code to be written in such a way that eventually re-execution of the cancelled instruction is initiated and execution of the original program is resumed. As an example of the use of this mechanism, suppose that the descriptor pointing to a segment is replaced by an escape descriptor. The effect will be that any attempt to access the segment in question will lead to an escape jump to a routine that might, for example, cause diagnostic information to be printed.

The last act of the routine before returning would be to reinstate the original segment descriptor and set a bit in PSR indicating that the instruction on which the escape jump occurred is to be re-entered.

The CALL instruction is used to make a full dress entry into a procedure using the stack. There is first a precall sequence executed in software; the call sequence itself is implemented in hardware. The CALL instruction is used in four ways. If the address is direct, or if it points to a vector descriptor or a code descriptor, a *normal call* takes place. The contents of PSR remains unchanged, that is there is no change in protection or privilege. If a call is made to a more trustworthy procedure (inward call) or to a less trustworthy procedure (outward call) it is necessary that the kernel of the operating system should be entered so that the values of ACR and PRIV can be changed. The address in the call instruction must therefore point to a system call descriptor. This has parameters which are interpreted by the kernel.

In all cases return is by means of an EXIT instruction. In the case of a normal call this can be via a *code descriptor*; the same is true for an inward call since the original values of ACR and PRIV can be picked up from the stack and reloaded by the mechanism of the EXIT instruction. A check is made that this reloading will not lead to an increase in privilege or protection status; this is necessary since the values of ACR and PRIV on the stack may have been overwitten in error at a level of privilege at which writing into the stack is permitted but not into PSR.

In the case of an outward call return must be via a system call descriptor. It is sometimes necessary for parameters to be passed from a higher ACR level to a lower ACR level. The VALIDATE instruction may be used at the lower level to check their validity. This instruction takes the value of ACR put on the stack at call time and compares it with the value of APF for the new segment. The result is indicated by condition code register (CC).

A call instruction may also be used to implement tasking. This is known as a task call and is via a system call descriptor with parameters that are interpreted appropriately by the kernel. Return is by means of another task call.

7. Slave stores

At the discretion of the designer of the hardware for a particular model in the range, small high speed slave stores may be associated with the main store so as to reduce the average access time; these are often referred to as buffer, or cache stores, and may work either on the associative principle, on the imaging principle, or on a combination of the two. The following are examples of slave stores that may be found in the various models of the 2900 series: instruction slave store; stack slave store; operand slave store. The instruction slave store can contain items only from the segment from which code is being

executed and the stack slave can contain items only from the stack segment defined by the SSN register. The operand slave store can contain items from any other segment. In addition the paging system is provided with the usual slave store designed to reduce the number of repeated references that must be made to the segment and page tables.

The slave stores servicing the main memory are operated in terms of virtual addresses rather than real addresses. It is, therefore, necessary for them to be cleared whenever a change occurs in the mapping from virtual addresses to real addresses, that is on a change from one virtual machine to another. Such clearing can be brought about as a side effect of certain instructions and it is unnecessary to go into the details here. An additional problem arises, however, in the case of shared segments, since an item in the main store may be updated under one virtual address while a slave store retains the old value under another virtual address. A solution to this problem would be to mark all shared segments to which writing access is permitted with a bit in the segment table, and to design the hardware so that items from segments so marked never find their way into a slave store. This, however, would be somewhat drastic and would reduce to an undesirable degree the efficiency with which the slave stores would operate.

Instead the items are allowed to pass into the slave store in the ordinary way, carrying with them the indicating bit, and advantage is taken of the fact that the writer of system software will, as a matter of course, protect any section of a shared segment that he needs to update by means of semaphores. It is a rule that he should make use for this purpose of the INCREMENT AND TEST instruction and the TEST AND DECREMENT instruction. These instructions, which are primitives at the hardware level, have the side effect of clearing from the slave stores items marked as coming from shared segments. The mechanism just described is capable of handling the problem that arises when two processors, each with its own operand slave (which may include a write buffer), write into the same area of memory.

A special instruction is provided for the purpose of clearing the address translation slave store of a processor other than that in which the instruction is executed. The specified processor remains halted until it receives a restart signal from the processor which halted it. This facility is available for use by the software when a processor makes a change to a segment table that is also being used by a process running in another processor.

8. Control of peripheral equipment

The principle is adopted in the 2900 range of relying on autonomous, asynchronous, peripheral controllers which are subordinate to the central processor. Input and output activities are always initiated by processes within the central processor. Inward connection (for example, of an online terminal) is achieved by an initial interrupt driven exchange which results in the establishment of a process in the central processor or the connection of the device to an existing process (for example, to a spooling process). A single input/output instruction is provided; the details of the required operations and storage addresses concerned are contained in data tables which are interpreted by the controller.

Since an input/output system cannot afford to wait while virtual to real address translation takes place before each store access, it is performed in the peripheral controller before the transfer begins. In fact the translation only needs to be done once, but clearly the main store area to or from which data is moved must not be paged out. It is therefore locked down in advance of initiation of I/O by the supervisor.

This it does by setting appropriate digits in the page or segment tables concerned. Slave stores whose contents have been invalidated by a peripheral input transfer will normally be cleared by the ACTIVATE instruction which restarts the process waiting to access the store area concerned.

There are four kinds of peripheral controller, each capable of driving a number of device controllers; these are:

General peripheral controller	—electromechanical devices
Disc file controller	—moving head discs
Sectored file controller	—fixed head discs or drums
Communications peripheral controller	—the communications subsystem

Each peripheral controller is assigned a *communication area* in the high speed store. This is established during system initialisation and contains (in fixed format) descriptors pointing to the control blocks which define the required input/output operations for the various device controllers. The communication areas and control blocks are accessible both by the central processor and by the controller concerned, access being controlled by means of a semaphore located in the first word of the area. The communication area also provides space into which termination responses are written by the controller. Synchronisation between the central processor and a controller is achieved by means of interrupts which, in the case of multiprocessor configurations, are broadcast to all central processors. The interrupts may either be taken as they occur or in a group after a preset time interval.

9. Considerations underlying the design

The 2900 series was designed against a background of falling costs of hardware relative to software. The provision of hardware features designed to support software could therefore be contemplated without misgivings. It was felt that the adoption of a stack architecture in the particular form described in this paper would, in addition to providing automatic dynamic allocation of workspace and local name space, facilitate the provision of the following features.

1. Hardware support for a procedure call and parameter passing mechanism.
2. Treatment of procedure calls and interrupts in a uniform manner as forced procedure entries which could be largely handled within the current stack and register environment, thus avoiding the unnecessary suspension and reactivation of processes.

Although in any particular model of the range the exact use that is made of slave stores is at the designer's discretion, slave stores had a definite place in the conception of the architecture. The architecture allows for the provision of local slaves fulfilling special functions in various places and indeed relies upon such slaves being provided for its efficient implementation. For example, it has been common practice in the past to provide the processor with a number of internal registers (accumulators, pointer registers, index registers), primarily to reduce the number of main storage access cycles needed below what would otherwise be necessary and thus increase the operating speed. The 2900 processor, on the other hand, contains a minimum of such registers, since an equivalent reduction in storage access time can be obtained by the provision of efficient slave stores. The reduction in the number of internal processor registers simplifies the task of the writer of compilers, since he is spared the problem of optimising their use.

The use of separate slave stores for separate functions enables any loss of efficiency occasioned by forced clearing of the slaves to be kept to a minimum. For example, the slave associated with the stack need only be cleared when the stack is switched.

Certain instructions in the instruction set have the side effect of bringing about the selective clearing of slave stores.

The protection system follows recently introduced practice of having a number of levels with efficient means provided for switching from one to the other. The hardware supports execute-only access as well as the more usual read-only and read-write access. Consideration was given to a more general protection system of the capability type, but it was felt that the benefits (aside from software and other problems involved in using such a system) were not sufficiently well established to justify its adoption. A conscious attempt has been made to provide a comprehensive set of internal interrupts some of which bring about the switching of the stack and some do not.

On the basis of this it has proved possible to design software that is tolerant of the occurrence of error conditions and other unscheduled events.

Acknowledgements

Whilst making no claim to have put forward any of the features outlined, the authors wish to record that the architecture was developed by many individuals over a period from proposals originally made by (amongst others): J. W. Bowthorpe, J. K. Buckle, J. Connett, (the late) R. D. Feather, D. Howarth, B. J. Moore, V. Pasquali, M. R. Patel, C. B. Taylor and M. R. Wetherfield.

They wish particularly to thank Professor M. V. Wilkes for his help and advice in the preparation of this paper. Finally, they record their thanks to ICL for permission to publish it.

Appendix 1 Registers in the image store

The number of bits in each register is given in brackets.

Visible registers

ACC	Accumulator (32, 64, or 128)
B	Index accumulator (32)
DR	Descriptor register (64)
LNB	Local name base register (16)
CTB	Cross-reference table base register (30)
PC	Program counter (31)
RTC	Real time clock (64)
SF	Stack front pointer register (16)

Invisible registers

IC	Instruction counter (24) (counts down by 1 for each instruction executed)
IT	Interval timer (24) (counts down by 1 each n microseconds, where $n < 16$ is given in the hardware manual for the model concerned)
LSTB	Local segment table base register (29)
PSR	Program status register (24); principal components are:
	ACR access control register (4)
	PRIV Privilege (1)
	OV Overflow (1)
	PM Program interrupt mask (8)
	ACS Accumulator size (2)
PSTB	Public segment table base register (29)
SSN	Stack segment number register (14)
SSR	System status register (31); principal components are:
	II Instruction incomplete indicator (1)
	RAM Real address node (1)
	PI Processor identification (2)
	EP Event pending indicator (1)
	DGW Diagnostic write (1)
	ISR Image store read (1)
	IM System interrupt mask (12)

Appendix 2 Operand addressing

An instruction consists of an operation code, a literal, and certain tag bits that indicate how the physical address to be presented to the access circuits of the store is evaluated. It will not be necessary to explain how instructions are expressed in binary form, but **Table 1** will show what addresses may appear in them.

DR stands for the content of the second word of the descriptor in DR; otherwise the names of the registers stand for their contents. N stands for the literal in the instruction. Brackets here indicate indirection. For example, ((LNB + N) + B), (TOS + B), and (DR + B) indicate items pointed at by pointers in descriptors held respectively in LNB + N, at the top of the stack, and in DR; '+B' indicates that the pointer concerned is modified by B.

Table 1 Addresses that may appear in instructions

	Access via descriptor		
N(literal)*	—	—	(DR+N)
(LNB+N)*	((LNB+N))*	((LNB+N)+B)	(DR+(LNB+N))
(XNB+N)	((XNB+N))	((XNB+N)+B)	(DR+(XNB+N))
(CTB+N)	((CTB+N))	((CTB+N)+B)	(DR+(CTB+N))
(PC+N)	((PC+N))	((PC+N)+B)	(DR+(PC+N))
TOS	(TOS)	(TOS+B)	(DR+TOS)
B	(DR)	(DR+B)	—

Instructions containing addresses given in the first five lines of the table require 32 bits (of which 18 are allocated to N), except that if N occupies less than eight bits, those marked with a star may be expressed in 16 bits. Instructions with addresses given in the last two rows of the table require 16 bits only. Instructions are also provided for performing operations on strings of consecutive bytes, these strings being addressed by vector or string descriptors held in DR and ACC.

Appendix 3 Interrupt classes

The interrupt classes are tabulated as follows:

Class	*Priority*	*Masking rules*	*Synch/ asynch*	*Stack switched/not switched*
1. System Error	1	—	A/S	SW
2. External	2	2	A	SW
3. Multiprocessor	3	2	A	SW
4. Peripheral	4	2	A	SW
5. Virtual Store		1	S	SW
6. Interval Timer	5	2A	A	SW
7. Program Error		1	S	SW
8. System Call		1	S	N
9. Out		1	S	SW
10. Extracodes		1	S	N
11. Event Pending		3	S	SW
12. Instruction Counter	6	2A	A	N

Priority: 1 is high

Masking Rule:
1. System error, if masked
2. If masked remains pending to system
2A. If masked remains pending to process
3. If masked ignored.

System error —Hardware detected errors and violation of Masking rule 1.

External —Interrupts from devices not having a connection to store.

Multiprocessor —Interrupts between processors sharing the same store.

Peripheral	—Interrupts from peripheral controller via the store access controller.
Virtual store	—Access requested to a legitimate page or segment which is not in virtual store.
Timer	—Interval timer.
Program error	—Interrupt due to illegal use of instructions or data.
System call	—Use of a system call descriptor in CALL or EXIT instruction.
OUT	—A software generated interrupt.
Extracode	—To allow the execution of selected instructions by software.
Event pending	—
Instruction counter	—Interrupt when IC goes negative.

NRSD 2.4.2. SPECIFICATION OF NEW RANGE PERIPHERAL INTERFACE

0 DOCUMENT CONTROL

0.1. Contents List

0.2. Status Record Details

Nil.

0.3. Change Proposals Approvals List.

All Approval Centres.

0.4. Documentation Cross References.

0.4.1. Functional Specification of General Peripheral Controller. NRSD 4.3.1.

0.4.2. Functional Specification of New Range Application Module. NRSD 4.4.2.

0.4.3. New Range Standards, NRSD 1.1.4.

0.4.4. Primitive Interface for Peripheral Controllers, NRSD 2.5.2.

0.4.5. Purchase Specification of 75 way connector. TD/IT/25.

0.4.6. Purchase Specification of Interconnecting Cable.7192103.

0.4.7. Primary & Secondary Power Distribution and Control Design Specification.DPEO.CSD.Technology Division. Document, TD/CEB 21.

0.4.8. Connection Assembly Drawing, 5068628.

0.4.9. Device Interface Design Specification. DPEO. CSD. Technology Division Document TD/CEB95.

0.4.10. British Standard Interface, BS 4421:1969

0.4.11. Switches and Indicators for New Range Equipment.7902174.

0.4.12. Maintenance Strategy. NRSD 2.3.13.

0.4.13. Coded Character Set, NRSD 2.11.1.

0.5. <u>Document Predecessors.</u>

Previous issue of this document was NRSD 2.4.2. Iss. 2/0.

0.6. <u>Document Acceptances.</u>

Nil.

0.7. <u>Changes Forecast.</u>

None.

1. GENERAL

1.1. Scope

This document defines the New Range Peripheral Interface Specification (NRPI) to be used by ICL for the connection of input/output peripheral devices to local and remote controllers. The specification does not apply to direct access devices.

1.2. Introduction.

This document is considered to include all changes that the design of equipment to this interface have shown to be required, together with those necessary to fulfill additional system functions that have been introduced since issue 1/2.

1.3. Changes.

Changes from NRSD 2.4.2. Iss.1/2.

i) Change of name to New Range Peripheral Interface, NRPI.
ii) Redefined electrical interface specification.
iii) Increased clarity of the termination of commands.
iv) Re-distribution of secondary and tertiary status bits.
v) Action to be taken on errors during a Sense command.
vi) Addition of device number to the property code bytes.
vii) Inclusion of assigned qualified device commands.

Changes from Iss 2/0 are:-

i) Change of connector type to MIL 28748 standard.
ii) Reduction in number of standard cable lengths.
iii) Improved definition of crisis time.
iv) Deletion of Change Masks and Modes command and addition of Connect command.
v) Change to allow Identify to be illegal if no interrupt is outstanding.
vi) No Operation and Identify added to list of commands which do not unset Tertiary status.
vii) Means of extending Attention interrupt types.
viii) Bit A2 in Attention byte is made device dependent.
ix) Reset time reduced to 1μsec.
x) Sundry changes to improve clarity.

2. GENERAL DESCRIPTION.

2.1 Definition

2.1.1. Electrically, the New Range Peripheral Interface is a symmetrical full-duplex interface with single character transfers controlled by a hand-shaking method. The defined procedures use the interface in a half-duplex mode and provide full 8 bit transparency over nine data lines.

The Interface provides for single channel multi-mechanism operation. The term means that whilst several mechanisms can be controlled simultaneously over one interface, data can only be interleaved on a block basis and not on a character basis.

2.1.2 New Range Peripheral Interface is defined as existing between two units X and Y which may be elements of a system with unit X existing higher in the hierarchy or closer to the system centre. Normally X is a general peripheral controller (GPC) and Y is a peripheral device.

Control of data transfers is assumed to be initiated from X and is achieved by X communicating with storage areas existing at a higher level than X in the hierarchy. In addition to communication with data areas, X is able to communicate with system software at a higher level.

Although initiated by X the rate of data transfers is under the control of Y.

2.1.3. The interface is divided into two symmetrical parts, each part providing data lines in opposite directions for the transfer of information between units X and Y. Additional lines provide a "hand-shaking" transfer mechanism for the information on the data lines.

2.2 Overall Operation

This description of the operations involved in the transfer of data is given in general terms. The rules governing such transfer are defined fully in later sections.
The implementation of particular actions by unit X is not fully described since this depends on the design of X and the system. In particular whether a function is carried out by hardware or software for example is not always covered in this description.

2.2.1 The General Peripheral Controller and Peripheral device raise and maintain their respective Operable lines at Logic 1 whenever their power rails are within prescribed tolerances. To avoid noise problems at switch-on, both units ignore all lines from the other unit until the incoming Operable line has been seen to be at Logic 1 for 1 msec.

2.2.2. The GPC raises its Acceptor Control line when it is prepared to accept an "interrupt" in the form of a Primary Status byte from the peripheral. The device raises its Acceptor Control line to be able to accept a command.

2.2.3. When first put under system control, i.e. in "auto", -see 7902174 the device sends over the Data lines to the GPC, a Primary Status byte with the Attention bit set. The controller responds to the Attention "interrupt" by sending an Identify Command to which the device replies with a mechanism address and a byte qualifying the "interrupt".

2.2.4. The device is now capable of functioning as and when required. When a command chain is started, the controller first sends over the Data lines to the peripheral an initialising sequence with the following format:

Initialise Command.
Mechanism Address
Command Mask
Status Mask
Mode Byte
(Limit Command)

2.2.5. After setting the defined masks and modes for the addressed mechanism, the device transmits a Primary Status byte to GPC. The byte contains bit significant information for the running of a command chain.

2.2.6. The controller sends a command to the device for the particular operation to be performed as required by the system.

2.2.7. The device obeys the command in the mode specified in the initialising sequence and if transfer of data is required either presents characters on the Data lines to GPC or accepts them on the Data lines from GPC. Transfer of all characters, including commands and status, is controlled by hand-shaking between the respective Source and Acceptor Control lines.

2.2.8. Termination of the command is signalled by the device sending Primary Status to the controller. Termination of one command is effectively a request for the next command.

2.2.9. Termination can be brought about either by the controller sending a 'Limit' command or by the device detecting for itself that it has reached a terminating point.

2.2.10. The Primary Status byte contains information which indicates whether the termination has been reached with or without unexpected events occurring. The resulting actions depend on the design of the controller and the system.

2.2.11. Generally a Sense command is sent to the device if unexpected events have been reported.
On receipt of the Sense command the device returns Secondary and Tertiary Status information which is examined by the system software to determine the appropriate recovery action.

2.2.12. If termination is reached without an unexpected event occurring the controller sends the next normal command.

2.2.13. The definition of unexpected events is defined for each device as those status conditions that are allowed to set the "Unsuccessful" bit in Primary Status by the Status Mask set in the initialising sequence.

2.2.14. Multi-mechanism operation is the only method of working.

2.2.14. contd.

with all commands applying to the mechanism addressed in the initialising sequence, null (hex 00) address for a single mechanism device.
Only one command can be in progress at any time due to command chaining so that commands not requiring transfers, (e.g. rewind,) are terminated when received, the action being carried out "off-line". Completion (of rewind) is signalled by setting the "Attention" bit in the next return of Primary Status. If transfers are in progress on another mechanism the next Primary Status will be at the termination of that transfer command; if not, primary status is returned immediately.
(Separation of the termination and attention "interrupts" is effected in GPC, see 6.5.3.3.)

2.3 Lines across the Interface

	Direction	Abbreviation
DATA FX 2^0 - 2^8 (9 lines)	XY	DFX
Parity FX	XY	PFX
Source Control FX	XY	SCFX
Acceptor Control FY	YX	ACFY
DATA FY 2^0 - 2^8 (9 lines)	YX	DFY
Parity FY	YX	PFY
Source Control FY	YX	SCFY
Acceptor Control FX	XY	ACFX
X Operable	XY	XO
Y Operable	YX	YO

Lines From unit X to unit Y have the suffix FX;
Lines From unit Y to unit X have the suffix FY.

2.4 Function of Lines across the Interface

The lines are named with reference to transfer of information between two units X and Y. Normally X is a General Peripheral Controller and Y a peripheral device.

2.4.1 Data FX 2^0 - 2^8 (9 lines)

These lines are used to transmit information from unit X to unit Y one character at a time under the control of Source Control FX and Acceptor Control FY. The information can be output data, device commands, masks, modes, mechanism addresses, etc, distinguished by means defined in section 6. DFX 2^8 is set to Logic 1 only for device commands.

2.4.2 Parity FX

This line is used to transmit a bit which makes the parity of the Data FX lines "ODD".

2.4.3 Source Control FX

This line is used to indicate that a valid character is on the Data FX lines. It must not be set unless Acceptor Control FY is set and once set must not be unset until Acceptor Control FY is unset.

2.4.4 Acceptor Control FY

This line is set by unit Y to indicate that it is ready to accept a character on the Data FX lines. Acceptor Control FY is unset to indicate that the character signalled by Source Control FX has been accepted. It must not be set again until Source Control FX is unset.

2.4.5 Data FY 2^0 - 2^8 (9 lines)

These lines are used to transmit information from unit Y to unit X one character at a time under the control of Source Control FY and Acceptor Control FX. The information can be input data, device status, property codes, etc distinguished by means defined in section 6. DFY 2^8 is set to Logic 1 only for device Primary Status, see section 6.5.

2.4.6 Parity FY

This line is used to transmit a bit which makes the parity of the Data FY lines "ODD".

2.4.7 Source Control FY

This line is used to indicate that a valid character is present on the Data FY lines. It must not be set unless Acceptor Control FX is set and once set must not be unset until Acceptor Control FX is unset.

2.4.8 Acceptor Control FX

This line is set by unit X to indicate that it is ready to accept a character on the Data FY lines. Acceptor Control FX is unset to indicate that the character signalled by Source Control FY has been

accepted. It must not be set again until Source Control FY is unset.

2.4.9 X Operable

This line is used to signal that unit X is operable and capable of transmitting or receiving information on the Data FX and Data FY lines under the control of the Source Control and Acceptor Control lines. The state of all lines from unit X is valid only when X Operable is set, see section 5.6.

2.4.10 Y Operable

This line is used to signal that unit Y is operable and capable of transmitting and receiving information on the Data FY and Data FX lines under control of the Source Control and Acceptor Control lines. The state of all lines from unit Y is valid only when Y Operable is set, see section 5.6.

3. PHYSICAL SPECIFICATION

3.1 Connectors

The connector to be fitted on all equipments using this interface shall be a 75 way connector to ICL Purchase Specification See TD/IT/25 Iss. 2.

3.2 Cable

3.2.1 The cable used shall be to ICL Purchase Specification 7192103

3.2.2 The cable will be available in the following standard lengths:

7.5 metres.
15 "
27 "
55 "

4. ELECTRICAL SPECIFICATION

4.1 General

This specification defines the electrical characteristics of an equipment to equipment connection system comprising 26 lines. Each line consists of a two core interconnecting cable, identified as core A and core B, driven by a driving circuit called the Transmitter, the output being detected by a circuit called the Receiver.

The Electrical characteristics are defined at the plug and socket interface in the equipment containing the Receiver, except where stated otherwise.

4.2 Definition of Signal

4.2.1 Logic States

The signal is defined as a voltage difference between core A and core B of a line.

Logic 1 on a line exists when core A is HIGH with respect to core B.

Logic 0 on a line exists when core B is HIGH with respect to core A.

4.2.2 High Level

High is defined as a positive voltage within the range 2.5 volts to 4.25 volts.

Outside these limits the signal is indeterminate.

4.2.3 Mean Level

The mean signal level with respect to the signal reference (see 4.6.1.1) for an alternating signal of unity mark-space ratio shall lie within 1.5 volts and 2.2 volts.

4.2.4 Propagation Delay

Propagation delays are measured at the point where the voltage difference between the two cores is:-

1 volt.

4.2.5 Transition Time

The signal transition time, positive or negative, as measured between 10% and 90% of the voltage difference waveform shall be not less than:

10 ns

4.3 Transmitter Specification

4.3.1 Function

The Transmitter shall be capable of generating signals as defined in Section 4.2 when connected in the specified manner, via cable and connector to a Receiver.

4.3.2 Pulse Repetition Period.

The minimum pulse repetition period which the Transmitter must be capable of generating shall be not less than 500 ns.

ie $f_{max} = 2\,MHz$

4.3.3 Minimum Pulse Width

The minimum signal pulse width, as measured at the points where the differential voltage exceeds 1 volt, shall be not less than 200 ns.

4.3.4 Limiting Terminal Voltage

4.3.4.1 A voltage within the range ± 8 volts with respect to zero volts from a source impedance of not less than 100 ohms, applied for a maximum period of 2 seconds in any one minute, to the output terminals of the Transmitter at Point X (see Fig.1) separately or simultaneously shall not cause damage to the transmitter or its associated circuits.

When the voltage is applied simultaneously there must be at least 100 ohms between each output and the generator.

4.3.4.2 Each transmitter output must be able to withstand a direct short circuit to the signal reference voltage (zero volts) for a period not less than 2 seconds in any one minute for a total of not less than ten times.

4.3.5 Removal of Power

Removal of power from the equipment containing a Transmitter shall not cause damage to any Receiver as specified in Section 4.4 which may be connected to this Transmitter, or its associated circuits.

4.3.6 Disconnection of Cable

Disconnection of the interface cable when power is present in both the equipments connected shall not cause damage to the Transmitter or its associated circuits.

4.3.7 Common Mode Noise Immunity

Definitions of Terms and Methods of Test are specified in ICL document No. / (to be issued).

4.3.7.1 High Frequency

The transmitter must maintain a signal voltage difference of not less than 200 mV at the receiver connector for any value of common mode voltage up to at least ± 8 volts in the frequency range 0.1 to 30 MHz. There must be at least 80 ohms between each output terminal and the common mode voltage generator.

4.3.7.2 Low Frequency

The transmitter must maintain a signal voltage difference of not less than 200 mV at the receiver connecter for any value of common mode voltage up to at least ± 2 volts for all frequencies in the range 0 to 0.1 MHz. There must be at least 55 ohms between each output terminal and the common mode voltage generator.

4.4 Receiver Specification

4.4.1 Function

The Receiver shall be capable of detecting a signal as specified in Section 4.2 and interpreting it as a precise logic state.

4.4.2 Pulse Repetition Period

The minimum pulse repetition period which the Receiver is required to detect is 500 nS.

4.4.3 Minimum Pulse Width

The minimum signal pulse width which the Receiver is required to detect is 200 nS.

4.4.4 Open Circuit Terminal Voltage

The open circuit voltage at either input terminal of a receiver shall lie within the range -8 volts to +8 volts, when loaded with a 10K ohm resistor to Zero Volts.

4.4.5 Input Impedance

4.4.5.1 The differential input impedance of the receiver shall be not less than 100 ohms and not more than 120 ohms at all frequencies from zero to 10 MHz.

4.4.5.2 The common mode input impedance of the receiver shall be not less than 50 ohms and not more than 75 ohms at all frequencies between 0.1 MHz and 30 MHz.

4.4.6 Removal of Power

Removal of Power from the equipment containing a Receiver shall not cause damage to any Transmitter as specified in Section 4.3 which may be connected to this receiver, or its associated circuits.

4.4.7 Disconnection of Cable

Disconnection of the interface cable when power is present in both equipments connected shall not cause damage to the Receiver or its associated circuits.

4.4.8 Noise Immunity

Definitions of Terms and Method of Test are specified in ICL document No.... (to be issued)

The receiver shall be capable of detecting a differential signal of 200 mV or less as a precise logic state in the presence of a common mode voltage of up to at least ± 10 volts in the frequency range 0 to 30 MHz.

4.5 Operable Line Receiver

4.5.1 Function.

The Operable line Receiver output shall be in the HIGH state for a transition on the input to logic 1 as defined in Section 4.2 provided the input remains at logic 1 for at least 2 m.sec.

The output shall revert to the low state if the differential voltage between core B and core A is greater than or equal to zero for at least 250 ns.

4.5.2. Open Circuit Terminal Voltage

This shall be the same as 4.4.4.

4.5.3. Input Impedance

This shall be the same as 4.4.5.

4.5.4. Removal of Power

This shall be the same as 4.4.6

4.5.5. Disconnection of Cable

This shall be the same as 4.4.7

4.5.6 Noise Immunity

Definitions of Terms and Method of Test are specified in ICL document (to be issued)

The Operable Line Receiver shall be capable of detecting a positive differential voltage between core A and core B of at least 200 mV as a logic 1 and a differential voltage between core B and core A greater than or equal to zero as a logic 0 in the presence of a common mode voltage of up to at least ± 10 volts in the frequency range 0 to 30 MHz.

4.6 Interconnection Requirements

4.6.1 Cable Screen and Earthing

4.6.1.1 The signal reference (Zero Volts) of the two equipments interconnected by this interface must be connected by the screen of the interface cable.

4.6.1.2 The cable screen must be a continuous screen between the Interface Connectors of the two equipments and must be electrically isolated from the screen of any other cable or any other metal work outside the two equipments. The surge impedence of the cable screen to true or building earth must be kept above 150 ohms on average. This can be achieved by keeping the separation between the cable and potential earth surfaces to 50 mm on average or greater.

4.6.1.3 The connection between the cable screen at the Interface connector and Zero Volts shall have the following characteristics.

Impedance 0.05 ohms maximum at frequencies from 0 to 1.0 MHz.

1.0 ohms maximum at frequencies from 1.0 MHz to 50 MHz.

4.6.1.3 (Contd)

Zero volts is defined for the relevant equipment by the requirements laid out in TD/CEB 21.

The connection must be made according to assembly Drawing 5068628.

4.6.2 Signal Connections

The signal connections between the Interface Connector and the Transmitter or Receiver circuit board shall be in twisted pair conforming to the following requirements:

4.6.2.1 Characteristic Impedance shall lie within the range:

100 ohms to 120 ohms.

4.6.2.2 The maximum length shall not exceed 1500 mm

4.6.2.3 Cable forming with lines carrying signals of amplitude greater than 5.25 volts or switching transients faster than 100 ns is not permitted.

4.6.2.4 The maximum length of any untwisted portion at the interface or printed circuit board connectors must not exceed 30 mm.

The rules for the signal line connections between the frame connector and the respective Transmitter or Receiver are laid out in TD/CEB 40.

4.6.3. Signal Termination Details.

4.6.3.1 The signal wires in the cable must be terminated at the connectors as detailed below:-

Twisted Pairs.	Core A.		Core B.			
Colours	Pin	Colour	Pin	Colour	Wire Identity	Code
BLUE/GREEN	1	BLUE	4	GREEN	DATA FX	DFX 2^0
BROWN/YELLOW	2	BROWN	5	YELLOW	"	DFX 2^1
BROWN/VIOLET	3	BROWN	7	VIOLET	"	DFX 2^2
BLUE/BROWN	11	BLUE	14	BROWN	"	DFX 2^3
GREY/RED	10	GREY	13	RED	"	DFX 2^4
GREY/YELLOW	8	GREY	12	YELLOW	"	DFX 2^5
BLUE/ORANGE	15	BLUE	18	ORANGE	"	DFX 2^6
BROWN/RED	16	BROWN	20	RED	"	DFX 2^7
WHITE/VIOLET	17	WHITE	21	VIOLET	"	DFX 2^8
GREY/VIOLET	22	GREY	25	VIOLET	PARITY FX	PFX
BLUE/GREY	23	BLUE	26	GREY	SOURCE CONTROL FX	SC FX
BROWN/GREY	24	BROWN	27	GREY	ACCEPTOR CONTROL FY	AC FY
GREEN/VIOLET	30	GREEN	33	VIOLET	X OPERABLE	XO
WHITE/BROWN	48	WHITE	51	BROWN	DATA FY	DFY 2^0
ORANGE/RED	52	ORANGE	55	RED	"	DFY 2^1
ORANGE/YELLOW	53	ORANGE	56	YELLOW	"	DFY 2^2
WHITE/GREY	54	WHITE	57	GREY	"	DFY 2^3
ORANGE/VIOLET	58	ORANGE	62	VIOLET	"	DFY 2^4
GREEN/BROWN	59	GREEN	63	BROWN	"	DFY 2^5
WHITE/RED	60	WHITE	64	RED	"	DFY 2^6
ORANGE/GREY	65	ORANGE	70	GREY	"	DFY 2^7
WHITE/GREEN	66	WHITE	71	GREEN	"	DFY 2^8
ORANGE/BROWN	67	ORANGE	72	BROWN	PARITY FY	PFY

4.6.3.1 (Contd)

Twisted Pairs.		Core A.		Core B.		
Colours	Pin	Colour	Pin	Colour	Wire Identity	Code
WHITE/ORANGE	73	WHITE	76	ORANGE	SOURCE CONTROL FY	SC FY
GREEN/GREY	74	GREEN	77	GREY	ACCEPTOR CONTROL FX	AC FX
GREEN/RED	75	GREEN	78	RED	Y OPERABLE	YO
GREEN/YELLOW	79	GREEN	80	YELLOW	SPARE (1)	
BLACK/BLUE	40	BLACK	43	BLUE	SPARE (2)	
BLACK/ORANGE	41	BLACK	44	ORANGE	SPARE (3)	
BLACK/GREEN	42	BLACK	45	GREEN	SPARE (4)	
BLACK/BROWN	36	BLACK	39	BROWN	SPARE (5)	
BLACK/GREY	35	BLACK	38	GREY	SPARE (6)	
BLACK/WHITE	28	BLACK	31	WHITE	SPARE (7)	
BLACK/RED	29	BLACK	32	RED	SPARE (8)	
WHITE/BLUE	34	WHITE	37	BLUE	SPARE (9)	
WHITE/YELLOW	46	WHITE	49	YELLOW	SPARE (10)	
ORANGE/GREEN	47	ORANGE	50	GREEN	SPARE (11)	

4.6.3.2 Spares need not be connected but care must be taken to prevent them shorting to earth or to each other.

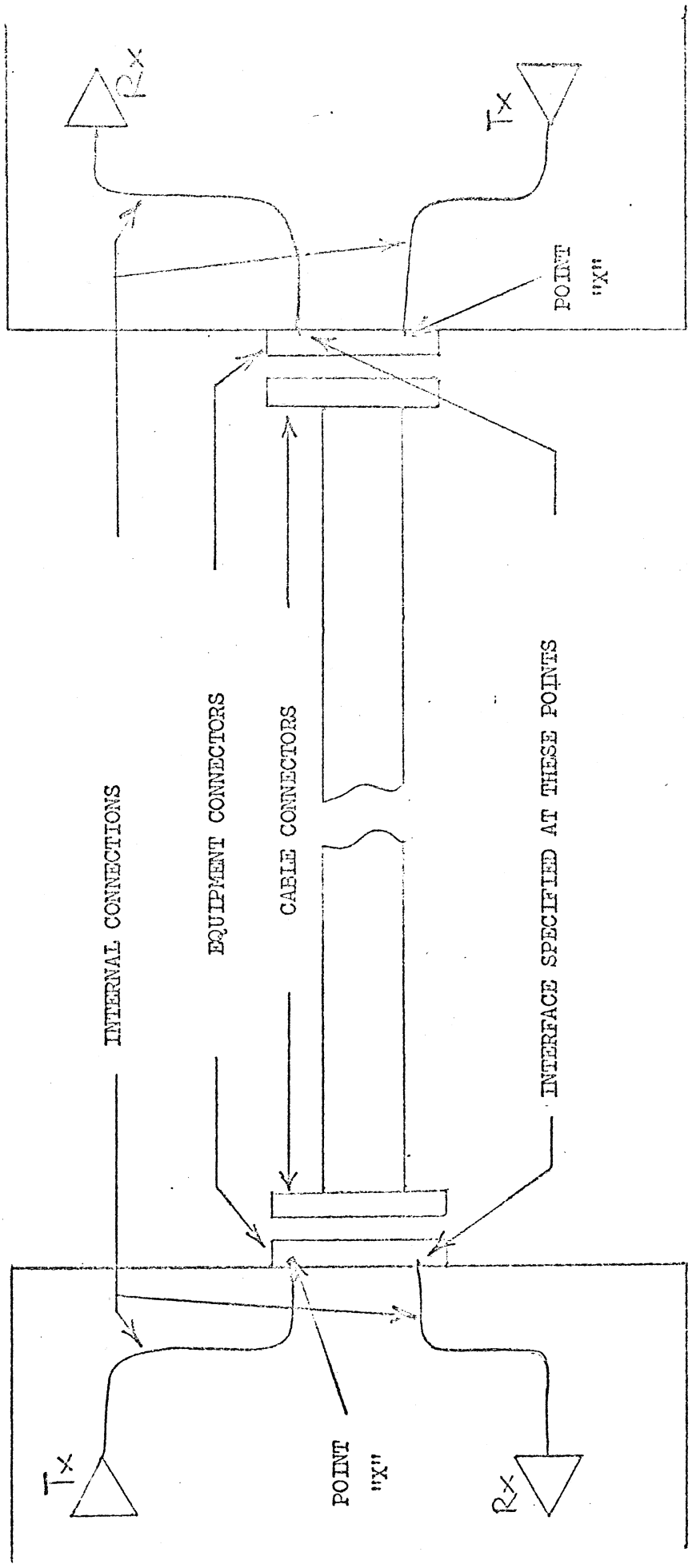

Rx
Tx
POINT "X"
INTERNAL CONNECTIONS
EQUIPMENT CONNECTORS
CABLE CONNECTORS
INTERFACE SPECIFIED AT THESE POINTS
Tx
POINT "X"
Rx

FIG. 1.

5 TRANSFER MECHANISM

5.1 Rules of operation of the interface lines

The rules of operation are defined for direction X to Y only, wi transfer in the other direction being controlled by the equivalent lines. The rules are based on those in BS 4421 and show no interdependence between the two halves of the interface. There are, however, certain logic interdependencies required by the procedures defined in section 6.

5.1.1 X Operable

5.1.1.1 A Logic 1 on the X Operable line indicates that unit X is operable and able to accept transfer of characters under the control of the Source Control lines FX and FY and the Acceptor Control lines FY and FX.

5.1.1.2 A Logic 0 on the X Operable line indicates that unit X is inoperable and the state of the other lines should be ignored by unit Y.

5.1.1.3 The X operable line should change from Logic 1 to Logic 0 only when the Acceptor Control FX and Source Control FX are at Logic 0. Transfers may be non-valid if the X operable line changes to Logic 0 at any other time.

5.1.2 Acceptor Control FY

5.1.2.1 A Logic 1 on the Acceptor Control FY line indicates that unit Y is ready to accept a character from unit X.

5.1.2.2 A Logic 0 on the Acceptor Control FY line indicates that unit Y is not prepared to accept a new character but has accepted the previous character, if any.

5.1.2.3 Unit Y must not accept data until it detects that the Source Control FX is at Logic 1. When unit Y has accepted a data character, it is permitted to set the Acceptor Control FY to Logic 0 at any time.

5.1.2.4 Unit Y must hold its Acceptor Control FY at Logic 0 until :-

(i) it has detected a Logic 0 on the Source Control FX line;

and (ii) it is ready to accept the next character.

It must then set it to Logic 1 for the next data transfer.

See section 6.11.3 for action at X Operable Reset.

5.1.3 Source Control FX

5.1.3.1 A Logic 0 on the Source Control FX line indicates that the signals on the Data FX and Parity FX lines may not be valid.

5.1.3.2 Unit X must hold its Source Control FX line at Logic 0 until :-

(i) it has detected a Logic 1 on the Acceptor Control FY line; and

(ii) the character on the Data FX and Parity FX lines is valid and must then set it to Logic 1.

5.1.3.3 Unit X must not change the character and parity until it detects a Logic 0 on the Acceptor Control FY line. When it detects a Logic 0 on the Acceptor Control FY line, it is permitted to set the Source Control FX to Logic 0 at any time.

5.1.4 Parity FX line

The Parity FX line must be used to transmit a digital signal to make the parity of the Data FX lines 'ODD'.

5.1.5 Data FX lines

5.1.5.1 Information is transmitted on 9 lines. A character can comprise any number of bits from 1 to 9, unused lines above the most significant bit must be held at Logic 0.

5.1.5.2 Each character is transferred from Unit X to Unit Y under the control of digital signals on Source Control FX, Acceptor Control FY, X Operable and Y Operable lines.

5.2 Transfer conditions

5.2.1 Unit Y must not rely on a specific time between the Data FX and Parity FX lines being steady and the Source Control FX line being set to Logic 1, so that it must, therefore, compensate for any differences in the delays in the signals in the interconnecting cable and its own receiving circuits.

5.2.2 The method of strict "hand-shaking" employed needs no timing rules, for the change in state of the Source Control FX line is conditional on detecting a previous change in state of the Acceptor Control FY

line and vice versa. The data transfer rate depends solely on the characteristics of the interconnected devices and delays in the interconnecting cables (See section 5.4).

5.3 Transfers from Unit Y to Unit X

Sections 5.1 and 5.2 apply for transfers between X and Y. An equivalent set of rules exist for the transfers between Y and X with the following change of line names.

For X Operable substitute Y Operable,
for Acceptor Control FY substitute Acceptor Control FX,
for Source Control FX substitute Source Control FY,
for Parity FX substitute Parity FY,
for Data FX lines substitute Data FY lines.

5.4 Transfer Rates

As stated in section 5.2.2 timing rules are not required for this type of interface. It is possible, however, to define the timing elements which determine the rate of operation. Since the interface is symmetrical it is sufficient to cover the factors for transfers in one direction only.

5.4.1 Figure 2 shows the transfer of characters on Data FX lines under the control of Source Control FX and Acceptor Control FY.

The solid lines show the time at the connector of unit X, the dotted lines at the connector of unit Y. Unit Y must introduce a delay at least equal to the cable skew and its own receiver skew before strobing Data FX with the leading edge of Source Control FX and unsetting Acceptor Control FY.

Similarly Unit X must introduce a delay to allow for its own transmitter skew to avoid setting Source Control FX before the Data FX lines and Parity FX line are steady.

5.4.2 In figure 2 , t_c = cable delay.

t_{Y1} = time unit Y takes to respond to SCFX changing to Logic 1.

t_{Y2} = time unit Y takes to request next character.

t_{X1} = time unit X takes to respond to ACFY changing to Logic 0.

t_{X2} = time unit X takes to obtain character in response to ACFY.

These response times include the delays through the interface receivers and transmitters in each case and in the de-skewing delays where necessary.

5.4.3 The minimum period for transfer of a character may be found by inserting values in the expression

$$4t_c + t_{X1} + t_{X2} + t_{Y1} + t_{Y2}$$

5.4.4 With similar hand-shaking logic at each end of the interface t_{X1} and y_{Y1} are the same and can be made short.

t_{X2} and t_{Y2} are dependent on the speed of units X and Y.

5.4.5 Using the transmitter and receiver characteristics specified in TD/CEB 95, and the cable characteristics specified in 7192103 the nominal theoretical transfer rates (for zero internal response times) for the standard cable lengths are as follows:-

Cable lengths.	
7.5 metres	2.0 . Mch/sec *
15 metres	2.0 Mch/sec *
27 metres	1.47 Mch/sec
55 metres	0.81 Mch/sec.

(* limited by the 500 nS minimum pulse repetition period, see section 4.4.2.)

Logic 1
Logic 0
Logic 1
Logic 0
t_{x2}
t_c
t_{y2}
t_c
t_{x1}
t_c
t_{y1}
t_c
DFX} must be valid.
PFX}
ACFX
SCFX

Figure 2.

5.4.6. (The maximum average transfer rate that a device is permitted to use may be further limited by the General Peripheral Controller and the higher system to which it is connected. Current designs are expected to be capable of handling an average rate of 0.8 Mch/sec. on the highest priority interface. Maximum instantaneous transfer rates are expected to be 1.3 Mch/sec. This does not preclude the possibility that controllers may be specified in the future with higher transfer rates).

5.4.7. Crisis Time.

Devices must be designed such that the time within which a request for transfer of a character to or from a controller requires to be serviced is not less that 1.5 times the device's nominal interval between character transfers or 10 usec, whichever is the greater.

This means that devices that have a nominal character time of 6.6 usec. and above, must provide at least one byte of buffer store. Devices with a nominal character time below 6.6 usecs. must provide sufficient bytes of buffer store to enable the 10 usec crisis time limit to be met, ie; for the number of bytes that could be transferred in 10 usec.

5.5. Time-out.

A device is not required to provide a timer, but is permitted to do so, to cover the possibility of failure of either the controller or device to comply with the hand-shaking rules of the interface. (Such a timer must be provided by higher parts of the system and could be based on failure to terminate a command chain in a prescribed time. Thus, although an interface might be frozen by failure to drop a Source Control in response to the lowering of an Acceptor control, or failure to drop an Acceptor Control in response to the raising of a Source Control, such a freeze would ultimately be detected by the higher system and could be corrected by generation of Reset, see section 6.11.)

5.6 Time-in

To provide protection against variation of signals on the interface control lines at switch-on, the controller and device must ignore all lines until the incoming Operable line has been seen to be at logic 1 for 2msec, see 4.5.1. All lines must be treated as invalid immediately whenever the incoming Operable line changes to logic 0, and remain so until it has been seen to be at logic 1 for 2 msec.

6 PROCEDURES

6.1. General

6.1.1. The procedures defined in this section are mandatory for the design of ICL in-house New Range equipment in which unit X is a General Peripheral Controller and unit Y a Peripheral Device.
This does not preclude the use of the electrical specification and transfer rules of the interface with other procedures for alien devices by mutual agreement of the system designer and the designer of the alien device.

6.1.2. Reference should be made to the description of the overall operation in section 2.2.

6.1.3. NRPI is specified as having nine lines plus parity in order to provide full transparency for the eight bit codes defined as the N.R. Standard.

6.1.4. The ninth Data FX line, DFX 2^8, is used uniquely to indicate that the other eight lines represent a device command and must not be used with any other meaning. Other control information from a controller to a device is distinguished by use of mandatory formats or by the use of prescribed control codes from the N.R. standard code set, NRSD 2.11.1.

6.1.5. The ninth Data FY line, DFY 2^8, is used uniquely to indicate that the other eight lines represent the device Primary Status and must not be used with any other meaning. Other status information and property codes from the device are sent, as data, only in response to specified device commands and are not distinguished on the interface

6.1.6. All codes across the interface in either direction, including those with DFX 2^8 or DFY 2^8 set to logic 1, are transferred in the same way under the control of the respective Source and Acceptor Control lines.

6.1.7 Bit numbering

In this specification, bits are numbered as defined in New Range Standards NRSD. 1.1.4.

Significance	Command or Primary Status.	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Bit number	-	A0	A1	A2	A3	A4	A5	A6	A7
DFX or DFY line	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

6.2 Initialising Sequence

This sequence is sent to set up a device prior to the starting of a command chain or sequence of command chains to the same mechanism. The mandatory standard format for all devices is:-

INITIALISE COMMAND. MECHANISM ADDRESS. COMMAND MASK. STATUS MASK. MODE. (LIMIT COMMAND.)

6.2.1 Initialise Command

This is one of the standard set of commands defined in section 6.4 and is sent with Data FX 2^8 set to logic 1. It must always be followed by the sequence defined in 6.2.

6.2.2 Mechanism Address

This consists of a single binary coded byte that contains the address c the particular mechanism on a multi-mechanism device to which the initialising sequence and all commands until the next initialising sequence are directed. The address codes must be defined in the device specification. The address byte is ignored by a single mechanism device. Invalid addresses must set Illegal address, bit A19 in Tertiary status, see 6.7.3.3.

6.2.3 Command Mask.

This consists of a single byte that specifies which of the maskable commands are to be responded to as illegal. Commands that can be masked must be defined in the individual device specification. See section 6.8.1.

6.2.4 Status Mask.

This consists of a single byte which specifies the groups of Tertiary status conditions that are allowed to set the appropriate bit in Secondary status and the Unsuccessful bit in Primary status. See section 6.8.2.

6.2.5 Mode

This consists of a single byte that specifies the mode of operation to be used by the device when executing all commands until a new mode is set in a new initialising sequence.

The significance of the bits in the mode byte must be defined in the individual device specification. It is intended for those properties

6.2.5 Contd.

which are file dependent. Receipt of an invalid mode must set Illegal mode, bit A17 in Tertiary status, see 6.7.3.3.

6.2.6 Limit Command

This is one of the standard set of commands defined in section 6.4 and is sent with Data FX 2^8 set to logic 1.

A device is permitted to terminate the initialising sequence without waiting for the Limit command after receiving four characters, by sending Primary Status as defined in section 6.9.2.2. In this case the Limit Command is not sent by the controller.

6.3. Device Commands

6.3.1. General

6.3.1.1. These consist of a single byte from the mandatory set of standard command codes defined in section 6.4. The four least significant bits constitute the basic command, the upper four bits being available for qualification of certain commands.
Data FX 2^8 is set to logic 1 for all device commands.
File dependent features should be set in the Mode byte sent in the initialising sequence and not by qualification of a particular command.
Use of any qualification or of any special command codes, must be approved by the design authority for NRPI. Approved codes are listed in Appendix 1 to assist in the selection of commands of similar meaning for different devices.
In general, by use of the Mode byte in an initialising sequence, see section 6.2., and qualification of standard device commands as described, no further qualification should be necessary. However, it is permitted to use further bytes sent as data after a standard command as qualifiers.

6.3.1.2. Due to the adoption of command chaining in New Range systems, a command must be terminated before a further command can be given to a device. The only exception to this is the Limit command which is only given when a command is in progress to bring about cessation of transfers. Termination is defined as the return of Primary Status with bit A3 set, see 6.5.2.4.
Commands cannot, therefore, be stacked by a device, e.g. to facilitate running at full speed. In some circumstances a device may require two interfaces in order to receive a command from another command chain whilst the previous command is still in progress.

Certain commands e.g. re-wind to magnetic tape units, should be terminated as soon as they are received, completion of the action being signalled by setting the Attention bit in the next convenient terminating Primary Status, see 6.5.2.3.

6.4 Device Command codes

6.4.1 Summary

Bit No. A	01234567
No operation	(1)00000000
Initialise	(1)00000001
Read	(1)XXXX0010
Write	(1)XXXX0011
Sense	(1)XXXX0100
Write control	(1)XXXX0101
Auto-load	(1)00000110
Auto-dump	(1)00000111
Disconnect	(1)XXXX1000
Connect	(1)XXXX1001
Identify	(1)00001010
Add to Command Mask	(1)00001011
Limit	(1)00001100
Add to Status Mask	(1)00001101
Send property code	(1)XXXX1110
Reserved	(1)XXXX1111

Notes:-

(1) All device commands are sent with Data FX 2^8 at logic 1.

(ii) All commands initiating transfer of characters on the Data FX lines from a controller to a device have bit A7 at logic 1.

(iii) All commands initiating transfer of characters (including primary status) on the Data FY lines from a device to a controller have bit A7 at logic 0.

(iv) Commands with bits A0 to A3 shown as X can be qualified. See Appendix 1 for assigned qualifications.

6.4.2 No operation

This command is used to obtain Primary status without effecting any change in the device.

6.4.3. Initialise

This command is used to set up a device at the start of a command chain. See section 6.2. for the information sent after the Initialise command. (The command is normally generated by the controller which then transfers four bytes of information from the Request Control Block of the command chain. A facility is provided for software to inhibit the sending of the initialising sequence for cases where the new chain is for the previously initialised mechanism with the same masks and modes).

(Privileged software is permitted to use this command provided bit A17 is set in the RCB of the command chain). A device is permitted to terminate the command autonomously after four bytes, otherwise termination is by Limit command.

6.4.4. Read:

This command initiates transfer of data from a device to a controller. Bits A0 to A3 can be used to qualify the form of reading e.g. read reverse.

For termination of reading, see section 6.9.2.

6.4.5. Write

This command initiates transfer of data from a controller to a device. In general, the data is that to be "written" on to a medium, c.f. Write Control, 6.4.7.

BitsA0 to A3 can be used to qualify the form of writing, e.g. (write) erase, write tape mark.

For termination of writing, see section 6.9.2.

6.4.6. Sense

This command initiates the transfer of a device's Secondary and Tertiary status. Termination can be brought about either by the controller sending a Limit command (as a result of expiry of the count in the Address List entry associated with the Sense command) or by the device having transferred all the available bytes.

All tertiary status bits must be unset after successful termination of the Sense command unless the causatory condition still exists. Only an interface parity error or an illegal command that occur during the execution of the Sense command must be allowed to set the Unsuccessful bit A0 in the Primary status sent to terminate the Sense command.

6.4.7. Write Control.

This command is used to transfer to a device, characters of a control type that are not in general to be written onto a medium.

6.4.7 contd.

Examples are format information to document readers and end codes to paper tape readers. Termination can be brought about by a Limit command or autonomously by the device.

6.4.8. Auto-load.

This command is provided for the transfer of data (programs) from a device into a cleared store. Receipt of this command must cause a reading device to read without an Initialise command and the associated address, masks and mode, on the lowest numbered and allocated mechanism. The device must automatically set a full set status mask and the appropriate mode. Reading must continue without further commands until a terminating point is reached that is defined in the individual device specification. A terminating primary status must be returned, unless an error condition occurs, only:-

(i) for card readers when all the cards loaded in a hopper have been read,

(ii) for paper tape readers, when an "end of Medium" code (1/09 in N.R. paper tape code) is detected on the tape;

(iii) for magnetic tape units, when end of block is detected.

Short block must not be set when terminating this command, see 6.5.2.5.

6.4.9. Auto-dump

This command initiates the transfer of data from a controller to a device in a continuous stream without an Initialise command and the associated masks and mode on the lowest numbered and allocated mechanism. The device must automatically set a full set status mask and the appropriate mode to be defined in the device specification. Termination will normally be brought about by a Limit command unless an error condition occurs. A device that is defined to accept the Auto-dump command must provide its own format effecting for writing the data continuously on to the medium.

6.4.10. Disconnect

This command releases a device from the Auto condition to the Manual condition in order to permit operator intervention. After sending a terminating Primary Status the device must set the "device in manual" bit A0 in Tertiary status, see 6.7.3.1.

6.4.10. Contd.

Disconnect can be qualified, see Appendix 1, to indicate that in addition to setting the device to Manual, an operator attention indicator must be set flashing. The command can be terminated by Limit command or autonomously by the device.

6.4.11. Connect.

This command is provided for use on communication equipment and may be qualified to signal connection at line or terminal level etc.

6.4.12. Identify.

This command is normally generated and sent by a controller only at the end of a command chain or immediately if no chain is running, in response to the setting of the Attention bit in primary status.

Privileged software is also permitted to send this command only if bit A17 is set in the RCB of the command chain.

The response to this command if the Attention bit A2 was set in the last return of Primary Status, see 6.5.2.3, must be the mechanism address followed by a single Attention byte, the bits c which define the reason for the Attention bit having been set, see 6.5.3. The Attention bit must be cleared only after receipt of this command and should only be set in the terminating Primary Status of the Identify command if an interrupt is pending on another mechanism. See section 6.5.2.3. for the specificactic of Attention Status and interrupts. Termination will normally be after a Limit command.

If the Attention bit A2 was not set in the last return of Primary Status, a device must either make the correct response o else terminate the command immediately with Illegal command set in Tertiary Status, see 6.7.3.3.

Identify must not unset Tertiary status, see 6.7.2.

6.4.13. Add to Command Mask.

This command is used to increase the range of commands to which the device is required to set the "illegal command" bit A16 in Tertiary status. A bit set to logic 1 in the single byte mask indicates that the appropriate command(s) as defined in the individual device specification must be treated as illegal. There is no requirement for bits that have been set in the mask by any previous commands to be set in the mask sent with the Add

6.4.13. contd.

Command Mask command. Termination can be by Limit command or autonomously by the device after one byte has been received.

6.4.14. Limit

This command is generated and sent by a controller to indicate that the current command must be terminated , see 6.5.2.4. and that no further characters are to be transferred over Data FX or Data FY lines for that command. Use of the Limit command is detailed in section 6.9.2.

6.4.15. Add to Status Mask

This command is used to increase the range of conditions that are to be masked, see 6.8.2. A bit set to logic 1 in the single byte mask sent with this command indicates that the associated conditions, as specified in the individual device specification, are to be allowed to set the appropriate secondary status bit and hence the Unsuccessful bit A0 in Primary Status. There is no requirement for bits that have been set in the mask by any previous commands to be set in the mask sent with the Add to Status Mask command. Termination can be by Limit command or autonomously by the device after one byte has been received.

6.4.16. Send Property Codes.

This command initiates the transfer to the controller of a number of bytes of information about the type, sub-device type, device number, etc., of the device to which it is addressed. Property codes are specified in Appendix 2. Termination will normally be after the device has transferred all available property codes unless a Limit command has been received.

6.4.17. Reserved command.

If a special command is required that does not comply with the requirements of being a qualification of any of the Standard commands. then the reserved standard command suitably qualified may be used, subject to the approval of the NRPI Design Authority.

6.5 Primary Status

6.5.1 General

Primary status consists of a single bit-significant byte and is transmitted on the DFY lines with Data FY 2^8 set to logic 1.
It is returned by a device under two circumstances:-

(i) to signal the termination of the current command, i.e. terminating status

and (ii) to signal that a defined event has occurred when no command is being executed, i.e. interrupting status. The interrupt may be unsolicited and must not require high priority servicing.

The device is permitted to combine the second type with the first type if a command is being executed at the time the interrupting event occurs. (Separation of the status types is carried out by the controller before passing them to the higher system. See NRSD 2.5.2.)
Standard meanings are defined for each bit in the primary status byte to control the running of command chains.

6.5.2 Assignment of bits

Using the N.R. standard nomenclature for bits, primary status is transmitted with Bit A0 on the DFY 2^7 line through to Bit A7 on DFY 2^0 line.

Bits are specified to have the following mandatory meanings.

Bit A0	Unsuccessful
Bit A1	Branch
Bit A2	Attention
Bit A3	Terminated
Bit A4	Short block
Bit A5	Long block
Bit A6	Condition X
Bit A7	Condition Y.

6.5.2.1 Bit A0 Unsuccessful

This bit must be set to logic 1 in the next return of primary status when a tertiary status condition has occurred that is masked by the current setting of the status mask. It is to be noted that 'unsuccessful' refers only to the conditions as defined by the mask. Status conditions that are not masked may be said to represent expected events for that particular process, whilst those that are masked represent unexpected events. Primary status must only be returned with bit A0 set to logic 1 when bit A3 is set to logic 1. (The precise use of this bit by the system is not defined in this specification, but, for information only, if set to logic 1 it results in the breaking of the current command chain. A Sense command is then sent to transfer Secondary and possibly Tertiary status into the main store for analysis).

6.5.2.2 Bit A1 Branch

This bit must be set to logic 1 as defined in the individual device specification for those devices that require to use it.
Primary status must only be returned with bit A1 set to logic 1 if bit A0 is set to logic 0.
(The precise use of this bit by the system is not defined in this specification, but, for information only, if set to logic 1 it results in a branch to the next but one LBE in the command chain.)

6.5.2.3. Bit A2 Attention

This bit can be set to logic 1 in any return of primary status independently of the other bits. It is used to signal events or conditions that do not require high priority servicing, i.e. no necessity to break a command chain, if one is running. Events and conditions that are allowed to set the attention bit must be defined in the individual device specification.

If an event occurs or condition arises when no command is in progress the device is permitted to return primary status immediately with only the attention bit set. This constitutes an unsolicited interrupt. If an event occurs when a command is in progress, the attention bit must be set when that command is terminated. The Attention bit is qualified by a bit or bits set in the Attention byte, 6.5.3. This byte is returned, together with the mechanism address on which the interrupt occurred, in response to an Identify command.

The attention status must only be cleared after the receipt of an Identify command and until this occurs, once the bit is set to logic 1 it must be set in each return of primary status when terminating any intervening commands, see 6.4.12.

If an unsolicited attention type primary status is being transmitted at the same time as a device command, the controller is required to accept the primary status but to discard it. The attention interrupt is not lost since bit A2 will remain set.

The peripheral is required to accept the command and action it, the type of termination being dependent on the tertiary status and mask at the time.

6.5.2.4 Bit A3 Terminated

This bit must be set to logic 1 when returning primary status to signal termination of the current command. Termination is defined as the point when the device is able to accept a new command. This will be at least after all character transfers have been completed, if any, but on some devices may be before all other device operations have been completed. Specification of the end of character transfers is given in section 6.9.2. (Although not defined in this specification, primary status with bit A3 at logic 1 and bit A0 at logic 0 generally results in the transmission of the next command from the command chain.)

6.5.2.5 Bit A4 Short block

This bit must be set to logic 1 in a terminating primary status for any command involving transfer of characters if the device detects that the end of transfers has been reached before a Limit command has been received from the controller. See section 6.9.2. If, when preparing to return status with Short block set, a Limit command is received, then neither Short block nor Long block should be set. Short block must not be set when terminating an Auto-load command.

6.5.2.6 Bit A5 Long block

This bit must be set to logic 1 in a terminating primary status for any command involving transfer of characters if the device receives a Limit command to end transfers before it has detected the normal end of transf as defined for that particular device or command. See section 6.9.2.

6.5.2.7 Bit A6 Condition X

This bit must be set to logic 1 in a terminating primary status if a particular condition as defined in the individual device specification occurs. An example of the use of this bit, which is provided to permit signalling of conditions without setting Unsuccessful, is Overflow on a printer.

6.5.2.8 Bit A7 Condition Y.

This bit must be set to logic 1 in a terminating primary status if a particular condition as defined in the individual device specification occurs.

6.5.2.9. Bits A4 to A7

(Although not defined in this specification, it is to be noted that these bits are compared in the controller against "ignore flags" that can be set for each command in a chain, as opposed to the normal status mask which applies to all commands in a chain. For example, commands in the same chain can define either to ignore long block or take account of it at the discretion of the program. See NRSD 2.5.2. section 5.8.)

6.5.3. Attention Byte.

6.5.3.1. The response to an Identify command received when bit A2 was set in the last return of primary status, is defined as being the address of the interrupting mechanism, null hex 00, for single mechanism devices, followed by an Attention byte.

Bits set to logic 1 in the Attention byte have the following significance.

Bit A0 Device set to Auto i.e. allocated to the system.

Bit A1 To be defined in the individual device specification.
Used on Magnetic Tape as Rewind complete.

Bit A2 To be defined in the individual device specification.

Bit A3 To be defined in the individual device specification.
Used on Magnetic Tape as Tape Mark found.

Bit A4 Device in Engineer state.

Bit A5 Device Available.

Bit A6 Device Not Available.

Bit A7 To be defined in the individual device specification.

Where a device has more types of Attention interrupt than can be contained in the single Attention byte, the excess qualifications should be put in Tertiary status and a bit defined in the Attention byte to indicate that a Sense command is required to read in the reason for the interrupt. Such bits in Tertiary Status should normally not be masked to cause Unsuccessful or else should be defined to be in bytes beyond the range of the 8 bit mask, see 6.8.2.

6.5.3.2. Bit AO. Device set to Auto.

A device is defined as being in one of two states providing that power is switched on. These are "manual" and "auto". Although communication is possible in the manual state, e.g. for the sending of Property Codes, the mechanism is directly under control of the operator and not the system. Change to the "auto" state, in which the mechanism is placed under system control, is signalled by the setting of bit AO and the sending of primary status with the attention bit set, see 6.5.2.3. (Reference should be made to the Standard for Operator's Switche and Indicators for New Range Equipment, 7902174 for further information on changes between the two states).
There is no similar requirement for signalling the change from the auto state to the manual.

6.5.3.3. Bit A4. Device in Engineer state.
This bit must be used by any device that requires to signal to the system that it is under engineer control. The setting of this bit should not itself be a reason for sending an interrupting primary status, see 6.5.2.3., the state being reported when some other interrupt, e.g. bit AO, is sent. (Information on the use of this bit and the use of a byte of Tertiary status for engineer communication with the system is defined in NRSD 2.3.13.)

6.5.3.4. Bits A5 and A6. Device Available/Not Available.
These bits must be used to signal to the system that the device should be added to or removed from the configuration list of available devices. An interrupt in the form of primary status with the attention bit set must be returned for any change from one state to the other, bit A5 or A6 being set, identifying the new state.
(In addition to the facility this provides of updating the configuration list directly from a device, the interrupts are also intended to provide a signal to system software of a change of level when input is suspended. This is based on the assumpti that the first input after a device is first "made available" is at system level. Thus, on breaking into job input a "not available" interrupt is generated followed by a "device availabl interrupt prior to starting input for a different job).
Further information on the generation of the interrupts is contained in 7902174.

6.6 Secondary Status

6.6.1 General

Secondary status consist of a single byte the bits of which have a one to one correspondence with the Status Mask. Bits have a standard meaning, each bit representing as far as possible those events that require a similar recovery action.

Each bit is related to one of the first eight bytes of Tertiary status. A bit set in the Status Mask allows the setting of the same bit in Secondary status whenever any bit in the related Tertiary status byte is set. Additionally the Unsuccessful bit A0 must be set in primary status.

Secondary status followed by Tertiary status is returned only in response to a Sense command. Both must be transferred over the interface as normal data characters with Data FY bit 2^8 at logic 0. Termination can be brought about either by use of a Limit command when sufficient bytes have been transferred or by the device terminating when all available bytes have been transmitted.

6.6.2 Assignment of Secondary Status bits

Bit A0 Mechanism Inoperable

Bit A1 Device error

Bit A2 Soft-ware error

Bit A3 Device Halted or Medium error

Bit A4)
Bit A5) Device dependent.
Bit A6)
Bit A7)

Status is transferred over the interface with Bit A0 on Data FY 2^7 line through to bit A7 on Data FY 2^0 line.

The assignment of bits is permitted to vary from the above by approval of the design authority for NRPI to allow for devices that require more bits for certain types of conditions.
It is mandatory that device specifications should conform to the assignment of status bits defined for other devices of a similar type.

6.6.3 Bit AO Mechanism Inoperable

This bit is set to logic 1, dependent on the Status Mask, when any bit is set in byte 0 of Tertiary status.
Events or conditions that are allowed to set bits in byte 0 and hence bit AO are those which, if occurring whilst a device is in use, i.e., crash stop conditions, are likely to involve the abandoning of the job or at least a major restart. In the presence of such conditions, starting a job would be inhibited by preventing a successful "setting to auto", except for "device in manual".
(It is to be noted that "inoperability" does not refer to the interface itself, this is signalled by dropping the Y Operable line).

6.6.4 Bit A1, Device error

This bit is related to byte 1 of Tertiary status.
Events that are allowed to set bits in byte 1 and hence if masked, bit A1, are those for which recovery involves a repeat of one or more commands, with or without operator assistance for restoration of media to a former position. The number of commands and transfers that may have to be repeated is device dependent. Events may be hardware malfunction, medium error, operator intervention, etc.

6.6.5 Bit A2. Software error

This bit is related to byte 2 of Tertiary Status. It is provided to indicate the receipt on the Data FX lines of a code, of any type, that has been defined as illegal in the individual device specification. Certain device command codes can be defined as illegal in the Command Mask at the discretion of the system software. Recovery is at the discretion of the software.

6.6.6 Bit A3 Device Halted or Medium error

This bit is related to byte 3 of Tertiary Status.
Events that are allowed to set bits in byte 3 and hence if masked, bit A3, are those for which recovery consists in eventually continuing from the stopping point, generally after an unsolicited attention primary status. Setting of any bits in this byte must normally be delayed until the current command has been completed and a successful terminating primary status has been returned. The next command will therefore be terminated unsuccessfully before any transfers occur.

Alternatively for devices that do not have conditions which comply with the requirements, this byte may be used for signalling errors additional to those in other bytes. In particular Magnetic Tape Units may use the byte to signal additional byte 1 type errors of a medium error category.

6.6.7 Bits A4 to A7. Device dependent.

These bits are related to bytes 4 to 7 of Tertiary Status.
They are intended

i) for device dependent conditions not covered by other bits

ii) extension of the number of bits available for reporting error conditions of the types in bytes 0 to 3.

iii) for reporting information about events that did not result in the loss of data but about which the system may require to be informed for logging purposes.

iv) for engineering diagnostic purposes.

v) to allow selective masking of individual Tertiary Status conditions from one of the other bytes.
Selective masking of an individual bit from a particular Tertiary Status byte can be achieved by specifying that a condition should set a bit in byte 6 say, as well as or instead of byte 1, to allow one type of hardware error to be masked whilst others are not masked.
Simple devices may restrict Tertiary status to a single bit in each byte. In this case, Tertiary status is equivalent to Secondary status and termination then occurs after transmission of Secondary status.

6.7 Tertiary Status

6.7.1 General

This may be any number of bytes, the first eight of which are related to particular bits in Secondary status. Bytes of Tertiary status are returned after Secondary status as a result of a Sense command until either a Limit command is received or until all bytes have been returned. Termination is signalled by returning Primary Status.

For conditions that are common to several peripherals specific bits must be used, otherwise the assignment of bits must be specified in the individual device specification.

6.7.2 Unsetting of Tertiary Status.

Tertiary status bits for which the causatory condition no longer exists must only be unset either by the successful termination of a Sense command or by the next legal command if not Sense, Identify, No Operation or Limit.

Certain types of status bits can be regarded as reporting a continuous condition and are unset only after the clearing of the condition.

The only conditions which are permitted to cause unsuccessful termination of a Sense command are Illegal command and interface parity error.

6.7.3 Assignment of bits in Tertiary Status

Bits are numbered sequentially through all bytes of Tertiary Status. The lowest bit number in any byte is transferred over Data FY 2^7 line.

6.7.3.1 Byte 0, Mechanism Inoperable

Bit A0 Device in manual. (See 6.5.3.2.)

Assignment of bits A1 to A7 must be defined in the individual device specification in accordance with sections 6.6.2 and 6.6.3

6.7.3.2 Byte 1, Device Error

Bit A8 Interface parity error. (See section 6.10.2.)

Bit A9 Further recovered interface parity before the first is cleared. (See section 6.10.2.2).

Assignment of bits A10 to A15 must be defined in the individual device specification in accordance with sections 6.6.2 and 6.6.4.

6.7.3.3 Byte 2, Software Error

Bit A16 Illegal command.

Set by receipt of a device command not specified for that particular device or currently declared illegal in the Command Mask, see 6.8.1.

Bit A17 Illegal mode. (See section 6.2.5)

Bit A18 Illegal data code.

Set by receipt of a character that is not legal for that particular device, e.g., non-graphic code to a line printer

Bit A19 Illegal address. (See section 6.2.2)

Assignment of bits A20 to A23 must be defined in the individual device specification in accordance with sections 6.6.2 and 6.6.5.

6.7.3.4 Byte 3, Device Halted.

Bit A24 Hold

Assignment of bits A25 to A31 must be defined in the individual device specification in accordance with sections 6.6.2 and 6.6.6. (Where byte 3 is used for Medium error, assignment must be defined in the individual device specification).

6.7.3.5 Bytes 4 to 7.

Bit A32 Recovered interface parity error. (See section 6.10.2.2).

Assignment of bits A33 to A63 must be defined in the individual device specification in accordance with sections 6.6.2 and 6.6.7

6.7.3.6 Additional bytes

If additional bytes above eight are required, e.g. for diagnostic purposes, this fact and the assignment of bits must be defined in the individual device specification.

6.8 Masking

6.8.1 Command Masking

The single byte command mask sent in an Initialising sequence applies to all commands until a new mask is set. The mask and the commands to which it refers must be specified in the individual device specification. The setting of a logic 1 in the mask defines to the device that the respective command(s) is to be regarded as illegal, see section 6.7.3.3.
The mask may be changed by the Add to Command Mask command, see 6.4.13, to increase the number of commands that are to be regarded as illegal. Masking cannot be reduced by this command.

6.8.2 Status Masking

The setting of any bit(s) to logic 1 in the status mask allows the same bit(s) to be set in Secondary status and also the Unsuccessful bit A0 in Primary status. The mask is first set in an initialising sequence and can be added to, but only to increase the number of conditions resulting in 'Unsuccessful', by the "Add to Status Mask" command, see 6.4.15, with which a mask is sent with logic 1 in the appropriate position(s).

6.9 Data

6.9.1 General

All characters except commands on the DFX lines and primary status on the DFY lines are regarded as data as far as the interface is concerned. Only the use to which any character may be put by a device or by the system determines whether it can be regarded as a true data character or a control character.

Certain applications as control characters have already been define as mandatory, e.g. masks, modes, secondary and tertiary status. Other applications are device dependent and may be subject to other range standards, e.g. format effectors, device controls, information separators, etc.

6.9.1. Contd.

It is to be noted that a form of multi-channel working is possible by defining multiple character transfer conventions, certain characters being treated as an address not recognised as such across the interface.

6.9.2 End of data transfers

The end of data transfers can be signalled from either side of the interface dependent on the device and the particular device command being executed.

6.9.2.1 Controller Limited

A controller can signal the end of data by sending a Limit command.

i) For transfers from controller to device the command is sent over the Data FX lines after the last data character has been transferred. The device must then return a terminating primary status as defined in 6.5.2.4.

ii) For transfers from device to controller, the controller must not raise Acceptor Control FX after accepting the last character that it is prepared to receive until it has detected that the Limit command has been accepted by the device. The device must then return a terminating primary status as soon as it is prepared to accept a new command.

6.9.2.2 Device Limited

Devices that can autonomously determine the number of characters to be transferred for any particular command either by counting or by detection of a defined end code are permitted to signal the end of data by transmitting a terminating primary status without waiting for a limit command which must not then be sent.

For transfers from controller to device, the device must not raise Acceptor Control FY after accepting the last character that it is prepared to receive until it has detected that primary status has been accepted by the controller.

6.9.2.3 It is to be noted that a device must raise its Acceptor Control FY after receipt of a command to transfer characters to the controller in order that a Limit command may be received. Similarly a

6.9.2.3 Contd.

controller must maintain its Acceptor Control FX at logic 1 on idle interfaces and after a command to transfer characters from controller to device in order to be able to receive primary status.

Logic inter-connections are required therefore between the two halves of the interface to facilitate the ending of the data transfers. The receiving end has priority control of the termination of transfers.

6.9.3 Transparency

6.9.3.1 New Range Peripheral Interface provides full 8 bit code transparency by the use of a 9 bit interface.

6.9.3.2 Where the procedures have to be implemented to a device designed to this interface via an 8 bit environment, as may be required for efficiency in communication systems, it can be accomplished by passing characters through logic which generates an 8 bit pre-fix code whenever Data bit 2^8 is set. This code is transmitted before device commands and primary status. When the pre-fix code is received it must be suppressed and Data bit 2^8 set when transferring the following character over the Peripheral Interface.

6.9.3.3 When using this method, since one code from the possible 256 codes is used as the pre-fix code it must not appear in data without special treatment, otherwise it will be interpreted as a pre-fix.

The method to be adopted is to transfer two such characters whenever one appears in data. An escaping mechanism will be necessary to double all pre-fix codes in data. Two in succession will result in four characters. A de-escaping mechanism must signal the last of any odd number of pre-fix codes as a genuine pre-fix.

The pre-fix code is chosen not to conflict with any legal primary status and will not be approved for use as a device command.

Bit A	0	1	2	3	4	5	6	7
Pre-fix Code	1	1	0	0	1	1	0	0

6.10 Parity Checking

6.10.1 Parity errors

Parity must be generated for each character that is transferred in either direction across the interface. To reduce the number of error recovery situations due to parity errors caused by noise, it is recommended that if a parity error is detected in any character strobed on the data lines, Acceptor Control should not be lowered and the character should be strobed again, since until Acceptor Control is lowered the character must be maintained on the data lines.

Recovery may therefore be possible in transient noise situations. Permanent errors resulting from hardware failure or a particularly noisy environment cannot of course be recovered in this way.

The use of multiple strobing, the period between strobes and the number of attempts are not defined in this specification but must be determined by the individual device designer (as a result of noise measurements and practical tests on the effectiveness of the technique.)

6.10.2 Parity errors detected by the Device.

6.10.2.1 Parity errors

Interface parity errors detected in any characters on the Data FX lines that are not recovered by multiple strobing must set the Tertiary status bit A8. If the condition is masked see 6.8.2, the command must be terminated immediately that the error is detected.

If a parity error is detected in a command, no device action should be initiated and the command should be terminated immediately. A character with DFX bit 2^8 at logic 1 can be assumed to be a command.

6.10.2.2 Recovered parity errors detected by the Device

Interface parity errors that are recovered by multiple strobing in commands or data codes must set a bit in Tertiary status, preferably bit A32. Independently of the masking of the bit, commands should be executed normally, the terminating status being dependent on the current status mask.

6.10.2.2 Contd.

If a further recovered interface parity error occurs before the bit set in Tertiary Status for the first recovered parity error is cleared, see section 6.7.2., the device must set bit A9 in Tertiary status.

6.10.2.3 Source Control FX line errors

Noise on the Source Control FX line can signal the presence of a character spuriously. The resulting action depends on the precise time that the spurious character is detected by the device and whether it appears to be a command i.e. DFX bit 2^8 set to logic 1 or a data character.

i) If whilst not executing a command the device detects, on the DFX lines, a character that is not a command (i.e DFX bit 2^8 not set) with correct or incorrect parity, it must reflect that character on the DFY lines without change, including incorrect parity, under the control of ACFX and SCFY. (This reflection feature is intended to be utilised in testing the interface).

ii) If whilst transferring data on the DFY lines, the device detects, on the DFX lines, a character with correct parity that is not a command (i.e. DFX bit 2^8 not set), it must set bit A8 in Tertiary status and terminate according to the current Status mask.

iii) A spurious character, detected on the DFX lines whilst a command to transfer characters from a controller to a device is in progress, is not directly distinguishable from a genuine character and must be treated on its merits.

6.10.3 Parity errors detected by the Controller

6.10.3.1 Parity errors

The controller is required to generate a Reset, see section 6.11, when it detects an interface parity error in any character on the Data FY lines that is not recovered by multiple strobing. (The controller also signals abnormal termination to the system).

6.10.3.2 Recovered parity errors detected by the Controller

The device is not made aware of interface parity errors that are recovered by multiple strobing in the controller. (The action to be taken by the controller is specified in the Controller specification).

6.10.3.3 Source Control FY line errors

Noise on the Source Control FY line can signal the presence of a character spuriously. The resulting action depends on the precise time that the spurious character is detected by the controller and whether it appears to be primary status or a data character.

i) If the controller detects on the DFY lines a character with DFY bit 2^8 not set, with correct or incorrect parity, whilst no command is in progress or after giving a command requiring output of characters to a peripheral, it is required to generate a Reset, see section 6.11. If DFY bit 2^8 is set, the controller should accept the character as a genuine primary status if its parity is correct and generate a Reset if its parity is incorrect.

ii) A spurious character detected on the DFY lines after a command to transfer characters from a device to a controller is not directly distinguishable from a genuine character and should be treated on its merits.

6.10.4 It is to be noted that devices and controllers can in some circumstances distinguish that a character is spurious by detecting either a) that the interface hand-shaking rules have not been strictly adhered to or b) that the logical rules between the two halves of the interface defined in section 6.9.2. have not been met. These are not mandatory requirements and must be defined in individual controller and device specifications if employed.

6.11. Reset

6.11.1 To achieve a reset of the interface the controller must set its X Operable line to logic 0 for at least 1 μsec
Whilst at logic 0, the device is permitted by the hand-shaking rules to ignore Source Control FX and Acceptor Control FX and can therefore set its Acceptor Control FY and Source Control FY to the initial state, i.e. ACFY at logic 1, SCFY at logic 0.
The device must immediately respond to the Reset by setting its Y Operable line to logic 0 for at least 1 μsec allowing the controller to set its control lines to the initial state, i.e. ACFX at logic 1, SCFX at logic 0.
(It is to be noted that the Reset can be initiated in the General Peripheral Controller by program.)

6.11.2 Reset can be used to clear the lock-out condition on the interface described in section 5.5.
It is also used to reset the interface after the detection by the controller of an interface parity error or spurious character on the DFY lines.

6.11.3 After receipt of a Reset the device must clear any command that may be in progress but must not return terminating prima status. Tertiary status bits that are set must not be unset and the device must stay in its original state, manual or auto, and await a new command.

The final state of a device if a Reset is received whilst an active command, e.g. reading or writing, is in progress, is indeterminate but may be defined in the individual device specification.
If a Reset is received whilst an unsolicited attention primary status is being transferred it is indeterminate whether the device repeats the interrupt after completion of the Reset.

Appendix 1.

Assignment of qualified and special device commands

1) Disconnect and Set MANUAL light flashing. 0001 1000

This command is a qualification of the standard Disconnect command and is used to draw the attention of an operator to take some action.

2) Engineer Write Control. 0001 0101

This command is used to transfer characters to be displayed on an Engineer's Peripheral Test Box. The normal request for data characters is inhibited and transfers are controlled by operation of a switch on the box. Termination is either by operation of a switch on the box or by Limit command. (Further details are defined in NRSD 2.3.13).

3) Send End Codes. 0001 1110

This command is a qualification of the Send Property Codes command and is used to establish the current end of block codes, primarily on paper tape equipment.

4) Read Reverse. 0001 0010

5) Skip to Tape Mark. 0010 0010

6) Skip Reverse to Tape Mark. 0011 0010

7) Skip forward 1 block. 0100 0010

8) Skip reverse 1 block. 0101 0010

9) Read forward with error correction. 0110 0010

10) Read reverse with error correction. 0111 0010

11) Erase. 0100 0011

12) Write tape mark. 0010 0011

13) Disconnect and Unload. 1101 1000

14) Rewind. 0011 1000

Commands 4 to 14 are used in magnetic tape systems.

Appendix 2.

A2.1 Property Codes.

No limit is set on the number of bytes permitted for encoding property codes but the assignment of the first three bytes is mandatory.

Byte 0. Type of device. See A2.2.1.

Byte 1. Device number. See A2.2.2.

Byte 2. Sub-device type.

Other bytes must be specified in the individual device specification and may include the following:-

i) Options available.
ii) Modes available.
iii) Current mode, etc., as set up by the operator.
iv) Parameters as currently set by program.

A2.2. Assignment of codes.

A2.2.1. Byte 0. Device type Codes (Mandatory).

Bit A	0	1	2	3	4	5	6	7
Paper Tape Punches.	0	0	0	0	0	0	0	1
Paper Tape Readers	0	0	0	0	0	0	1	0
Card Punches	0	0	0	0	0	0	1	1
Card Readers	0	0	0	0	0	1	0	0
Magnetic Tape Units.	0	0	0	0	0	1	0	1
Line Printers.	0	0	0	0	0	1	1	0
Graph Plotters.	0	0	0	0	0	1	1	1
Visual Display Units.(including OPER 1)	0	0	0	0	1	0	0	0
Graphic Display Units.	0	0	0	0	1	0	0	1
Document Readers.	0	0	0	0	1	0	1	0
Typewriters.	0	0	0	0	1	0	1	1
Communications Terminal.	0	0	0	0	1	1	0	0
Switching Unit.	0	0	0	0	1	1	0	1

Other codes will be assigned as required and will apply to other New Range equipment not necessarily designed to this interface specification.

A.2.2.2. Byte 1. Device Number.

This byte must be used as an eight bit device number, coded as two hexadecimal digits, by which the device is known by the operators and the "supervisor", (operating system). Normally, only numeric values, in the range 0 - 99 are used in conjunction with a pair of alpha characters, derived from the first byte of the property codes, indicating the type of device, e.g. LP05, line printer No.5.

The device number may be prewired but must be capable of being set to any value by an engineer on installation or on site enhancement.

Least significant bits are to be used for the mechanism number on a multi-mechanism device.

REFS TO SERIES 39 FROM COMPUTERGRAM 20/09/90

ICL_CLAIMS_MIPS_LEADERSHIP_IN_THE_MAINFRAME_
WORLD_WITH_SERIES_39_SX_-_WITH_MORE_TO_COME

Publication Date PD= 04/05/90
Document Number DN= 1419 001
Text TX=
"These are not IBM attack machines", remarked ICL chairman Peter Bonfield yesterday as the UK mainframe maker launched two new systems at the top of its VME-based Series 39 that look like the company's answer to IBM's forthcoming Summit machines and are claimed to use the most powerful general purpose mainframe uniprocessor currently available. And his UK managers sharing the platform went on to use IBM's 3090 Series as the price-performance yardstick against which the new SX systems were judged. The SX 580-20 and SX 550-20 systems have been under development since the launch of the original Series 39 five years ago, and ICL has spent something in the region of #200m to do the job. They are built using Fujitsu's Hawk Emitter Coupled Logic arrays, the same base technology that is used by Amdahl in its two-year-old 5990 mainframe. Fujitsu fabricates the circuits to ICL's design, and ICL assembles the things on its own 42-layer boards. The first two models, the SX 580-20 and SX 550-20 are dual processors, and the company is planning to go up to six processors. Uniprocessor models will follow next year. A two-node SX 580-20 is rated by the company at 90 IBM MIPS, which is around 60% more powerful than existing four-node Series 39 Level 80s, or equivalent to the performance of a four-processor IBM 3090/400 J. The SX550-20 in a two-node configuration comes in at 60 IBM MIPS, whilst a six-node 580-20 is expected to deliver up to 300 MIPS. Main memory on the new systems goes up to 512Mb per node in units of 128Mb, and over 100 FDS 5000 disk drives can be attached, allowing up 6Tb of storage to be accessed. A new 200Mbits-per-second Macrolan 200 optical fibre link is also introduced, enabling SX nodes to be placed over a mile apart. The flat CPU board carries 336 of the ECL chips, each chip having 3,000 logic gates. Input-output traffic flows through up to 16 50Mbit-per-second Macrolan links on each node - which again can be over a mile apart - though extended versions of Macrolan allow disaster-tolerant sites to have devices attached up to 15 miles from the node. Local area network connection is via a 10Mbit-per-second OSLAN highway, wide area networking is provided by the Series 39 X25 controller. Prices go from #5m for a two-node 580-20 with 256Mb RAM, (#325,000 per quarter under ICL's exchange hire), and #3.5m for a 550-20, (#215,000 per quarter), up to #15m or more for a six-node 580-20 implementation. Although there are no plans to offer a native-mode Unix operating system on the Series 39, ICL is currently working on an implementation of X/Open Co Ltd's Common Application Environment interface which will be offered on the line next year, enabling compliant applications to be transferred between VME and Unix environments. ICL claims #20m in advance orders for five two-node systems. In the UK, the Inland Revenue is installing a Series 39 SX 550-20 its Telford development centre, and an SX 580-20 which will take over the workload of a four-node Series 39 Level 80 system. And Yorkshire Electricity is replacing five Series 39 Level 80s with two SX 550-20s at its Leeds office. In France, OR Telematique SA has ordered an SX 580-20 to handle the expansion of its on-line business information services. It currently uses a two-node Series 39 Level 80 at its Loire Valley chateau. ICLograms - p2

I C L O G R A M S

Publication Date PD= 04/05/90
Document Number DN= 1419 005
Text TX=

All this talk about partners for ICL rather misses the point that the picture has changed dramatically since the concept was first mooted by STC Plc, and these days, among bid European computer manufacturers, ICL is the one that needs a partner least: there have been talks with both Siemens AG and Ing C Olivetti & Co SpA, although it is not clear that anything is on-going, but ICL acknowledges that it approached Nixdorf Computer AG and proposed a combination before the force majeure from Munich rendered all other bets off; it also sought a stake in Ingres Inc, but does not now feel it needs one - but is rather pleased that DEC is about to take a stake as it thinks it gives its preferred relational database manager increased visibility and credibility in the market.

i c l

One way and another, mainframes may now account for less than 50% of ICL's total turnover but the new Essex - spell it SX now the machines are out - represent a triumph of British development and engineering, albeit exploiting Japanese technology: point is that ICL appears to be exploiting Fujitsu's technology rather better than Amdahl Corp if its MIPS ratings are to be believed, because although the new SX CPU is clocked at 12nS where Amdahl's 5990 CPU ticks over at 10.8nS, ICL reckons its getting 47 MIPS out of its CPU, a little more than Amdahl, but crams the CPU into far fewer cubic feet - much of that of course is the Godawfulness of the 370 architecture that Amdahl somehow has to get to grips with.

i c l

Amdahl Corp's problem, enunciated the other day, that its profits are going down as it ships more 5990s and fewer 5980s because the latter are more expensive to other mainframer - one of the points missed by the City is that ICL's research and development is so much more cost-effective than that of its US competitors - and likely the continentals and the Japanese as well: it's simply cheaper to do it in Britain.

i c l

Yes but why does ICL need machines of this performance - the first time in its history when it felt able to claim world leadership in uniprocessor MIPS? The answer is to reassure its existing customers that there will be a growth path when they start to run out of gas. i c lWho are these customers? Well the first five are named on today's front page, and ICL says that its biggest customers' hunger for MIPS is growing at 40% a year - which makes the company confident that it can grow its mainframe business at just over 10% a year for the next several years where the mainframe business as a whole is likely to grow at between 9% and 10% a year.

i c l

Just how good is the new machine? ICL lines it up against the IBM 3090-400J and the Amdahl 5990-1100 and reckons that the four processor IBM box delivers 81 MIPS, weighs 14,000Kg, covers 20 square metres, takes 88KW and throws out 80KW as heat; the Amdahl gets 91 MIPS out of three processors, weighs in at 7,910Kg, covers 14 square metres and takes 40KW, throwing out 37KW as heat; as for the SX 580-20, ICL says it delivers 90 MIPS, weighs just 1,200Kg, covers 3 square metres, takes in 15KW and throws 14KW back - sorry users, you're not going to be able to keep your office block warm in the winter on that.

i c l

The operators' union will
be picketing ICL's headquarters when this gets out, too - the
company did a survey of its major sites here in the UK, and
found that on average, a VME site required only half as many
operators as an MVS site, which is pretty important at a time
when the people that feed and water the mianframe are so hard to
find. i c lOther snippets on the ICL mainframe front include the
fact that 70% to 80% of sites are using the company's unique
CAFS Content-Addressable File Store - but then the thing does
come bundled with the box.
And the idea of converting to
VME is now taken as normal practice - over 70% of ME29 sites
have now converted to VME on the Series 39 (though it's said
that there are still two or three George 3 users).

i c l

Wilmot moves ICL more into system building

by Kevin Cahill

NEW policy directions at ICL aim to convert the company away from being a manufacturing-based organisation into a systems builder and distributor having strong software expertise and only some manufacturing capability.

A briefing given by ICL chairman Christophor Laidlaw and managing director Robert Wilmot to stockbrokers in London last week heard Wilmot outline his plans for co-operation with foreign companies and an implied further slimming of manufacturing in the UK.

The briefing is the latest at which Laidlaw and Wilmot are seeking to restore financiers' confidence in ICL.

According to those attending, Wilmot's plans are:

- The ending of all mechanical and electromechanical production of peripherals and associated equipment. ICL currently makes some tape drives, terminals and printers at Letchworth, where 500 redundancies have already been announced.
- Agreements to market other companies' machines through collaborative ventures. Brokers were left with the impression that a link with Fujitsu in Japan for the supply of computer and telecommunications equipment is close.
- Ending involvement with semiconductor development, though probably retaining some design capability.

Some brokers felt that the future of Logiclayer, ICL's printed circuit board production facility which also markets to outside organisations, was in doubt. But an ICL spokesman said this was a misinterpretation.

Wilmot told the brokers that £3 million saved by the decision to end the VME/K operating system is to be committed to the development of Information Processing Architecture, ICL's answer to IBM's Systems Network Architecture.

All these ingredients mark the culmination of a whirlwind review of ICL's strategy undertaken by the new managing director over recent months. His first aim, he told the meeting last week, is to enable the company to survive, the second to return it to profitability, and the third to restore real profitability. Some £50 million has already been cut from ICL's direct costs, he said.

On the financial side, those aims were boosted by near unanimous backing from an Extraordinary Shareholders Meeting held in London on Friday which approved the capital restructuring that has already taken place.

The meeting was told that Murray Stuart, deputy managing director for finance and administration, has resigned and will leave to join Metal Box on November 30. As

● **Turn to back page**

STUART . . . resigned

Change of emphasis at ICL

● **From front page**

finance supremo Stuart has been under increasing criticism from the City over ICL's financial position.

Wilmot told the assembly of brokers and analysts that he has been in the US, Canada, Japan and Europe negotiating collaborative agreements to market other manufacturers' machines. The first to come to fruition is likely to be with Three Rivers in the US for its Perq 16-bit microcomputer system. But a much more important one is likely to be with Fujistu, for the supply of computer and telecommunications equipment.

ICL deputy managing director Peter Ellis is just back from Japan, and sources at last week's meeting said an agreement with Fujitsu is probably less than eight weeks away.

Though ICL already has a technology agreement with Hitachi, it seems to have gone to Fujitsu because there are already three companies selling the Hitachi range in Europe. All three – Olivetti, BASF and National Advanced Systems – have been marketing aggressively in Britain and Europe.

Fujitsu by contrast has only one European agreement – with Siemens. It has close ties with Amdahl, and takes on the manufacture of the US company's mature products.

These links should not conflict with ICL, which has already stated it does not intend to stay in the giant mainframe business.

SUNDAY TIMES 6 SEPTEMBER 1981
business news

Revealed: ICL's survival plan

by Richard Brooks

ICL is to collaborate with US and Japanese companies as part of its master-plan designed to steer the ailing British computer firm safely through the 1980s.

The plan is the result of a three-month whirlwind review of the company by new managing director Robb Wilmot, who was brought in when ICL was on the brink of being taken over by Sperry Univac. Also in the pipeline are a new range of mini-computers; the phasing out of very large mainframe computers; and the ending of peripheral production equipment.

A week tomorrow Wilmot will spell out details of ICL's first collaborative venture. It will be with the Pittsburg-based Three Rivers Computing Corporation, one of the rising stars among American mini-computer firms.

The deal, which means ICL making under licence and marketing Three Rivers computers in Europe, will get ICL involved in the crucial "office of the future."

The Three Rivers product, known as Perq, is a very powerful and versatile mini computer, which is ideal as a work station and for what are known as local area networks. These connect a number of small computers on a single site, together with files and printers, so that they can "talk" to each other and to outside networks.

The Perq, which will be marketed as the ICL Perq in Europe, already links up with major office network systems such as Xerox's Ethernet. Frank Williams, Three Rivers vice-president (sales), said the Perq is ideally suited for office automation. "For Three Rivers the deal also gets us into Europe, where at present we just have tiny sales," he added.

ICL's next collaboration is likely to be with Fujitsu, one of the giants of Japanese computing. Wilmot was in Japan five weeks ago, and ICL deputy managing director Peter Ellis was there at the end of August. The deal, which should be announced within two months, will mean ICL buying the Japanese firm's chip design and manufacturing technology, which is renowned throughout the world.

In return ICLI will market Fujitsu machines in Europe. At the moment the Japanese firm has only one European arrangement, with Siemens, the German computer and electronics giant.

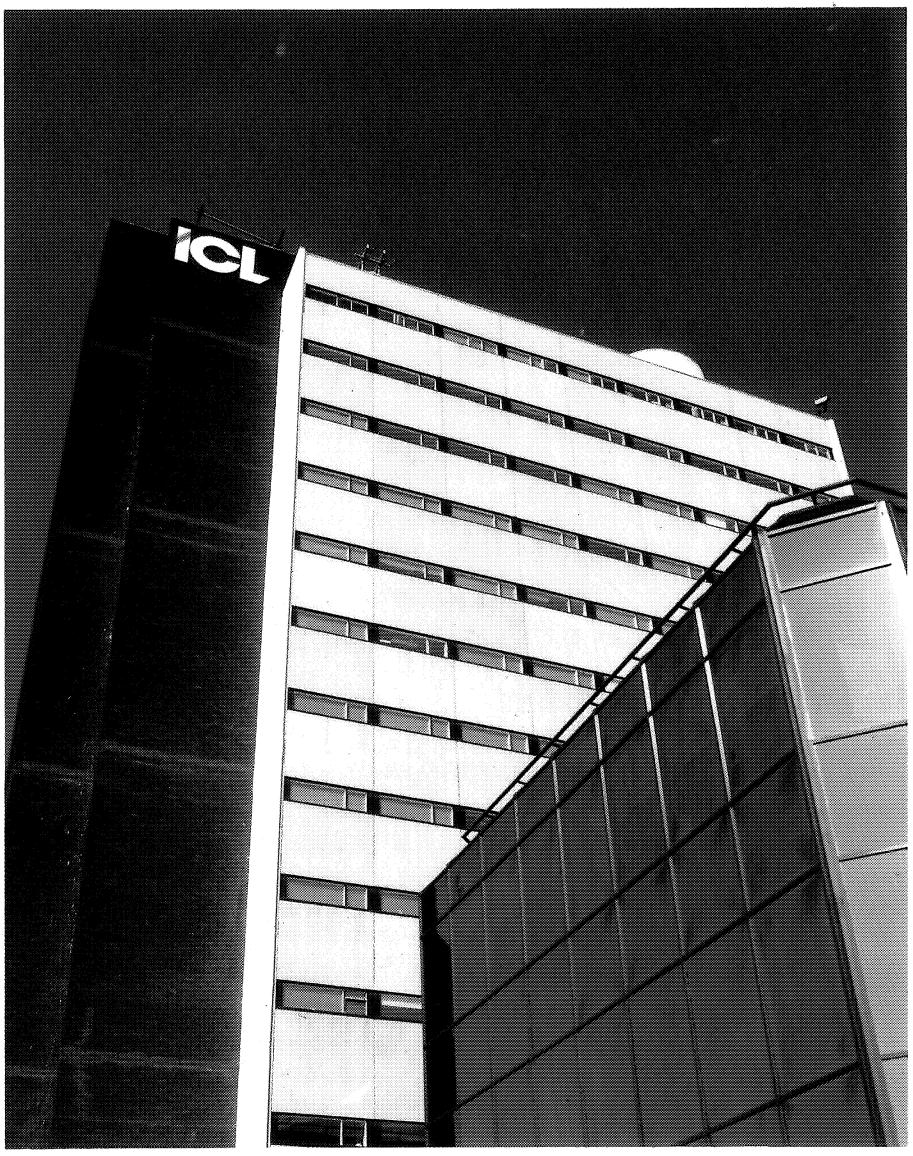
ICL has also been talking to Fujitsu about supplying telecommunications equipment. The third partner in the discussion is believed to be British Telecom, with talks centring around satellites. BT already has plans for a European satellite starting in 1983. More computer firms are also getting involved in satellites, such as IBM's joint venture in the United States with Comsat and Aetna.

Next month ICL will announce a major new series of mini computers to take over from its ageing range of minis, the 1500 series. The new computer will be ICL's smallest yet, and heralds its move into the upper end of the micro market, though not as small as personal computers. The new computer will be ideal for use in network communications.

ICL will also be ending its 2970 and 2980 range of very large main frame computers. These computers, which are mainly for governments and nationalised industries, have not sold well. Instead, ICL will concentrate on its smaller and more successful mainframe computer, the 2966, giving it add-on capabilities for more power and uses. ICL is also closing down all production of peripherals, such as tape drives and printers.

MAINFRAME SYSTEMS

ICL Mainframe Systems designs, develops and markets a complete range of systems for corporate computing. Creativity and innovation are the keys to our success.



ICL

AN STC COMPANY

Welcome

ICL — the international Information Systems company — designs, manufactures and markets an extensive range of computer systems and services together with supervisory and network management software.

The STC Group, of which ICL forms the major part, has a turnover in excess of £2.3 billon p.a. and employs over 33,000 people worldwide.

Within this highly successful Group, ICL produces a turnover in excess of £1.3 billion p.a., employs over 20,000 people, operates in more than 70 countries and supports over 80,000 installations throughout the world.

The ICL Group is made up of five divisions — International, Europe, UK, Retail and Product Operations. The Product Operations Group includes the manufacturing, development and central marketing organisations for Office, Network, Cross Range Software & Services and **Mainframe Systems.** It is Product Operations' responsibility to ensure that ICL's product range is in line with market needs and provides the very best platforms for business solutions to ICL's customers around the world.

ICL Mainframe Systems is a crucial part of this success and provides over half of ICL's annual turnover.

In providing a service to its customers, ICL has focused on the three areas of corporate, departmental and personal computing. Corporate systems are those concerned with the strategic operations of a business. Departmental systems support the work of teams within a business. Personal systems satisfy the computing needs of the individual. Mainframe Systems is the supplier of central and distributed corporate computing solutions.

Over 1300 people are employed by Mainframe Systems and the majority hold degree or equivalent formal professional qualifications. A substantial number have been with ICL for many years thereby benefiting from the excellent career and travel opportunities within the Company. ICL is very keen to ensure that skills are fully developed and is a leader in providing part time and home based working to secure the skills it requires. Currently Mainframe Systems employs over 100 people working from home in this way.

Mainframe Systems is based in West Gorton, Manchester and is a highly regarded member of the local community, in particular supporting many local events. ICL was the winner of the 1988 North West Business and Industry Award for its commitment to the North West.


ICL
ICL
ICL
ICL

Products

ICL Mainframe Systems is responsible for the design, development and validation of an extensive range of hardware, software products and services. These products are targetted at several distinct markets in both existing ICL and competitive computer sites. These markets include large corporate data processing sites, highly resilient, high throughput terminal servers and information centres, together with distributed systems and free-standing or networked departmental applications.

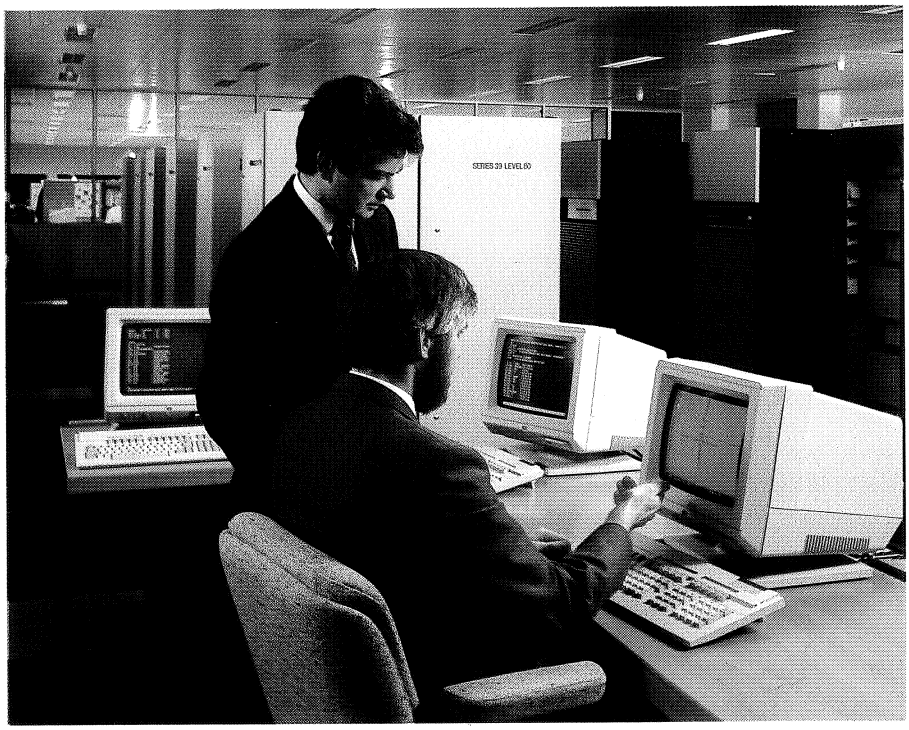
The ICL Series 39 is specifically designed to meet the needs of this continually expanding information systems market.

Mainframe Systems won the Queen's Award for Technological Achievement in 1988 for the design of Series 39, in particular the nodal architecture and the world's first use of fibre optic cables for high speed interconnection of processors and peripherals.

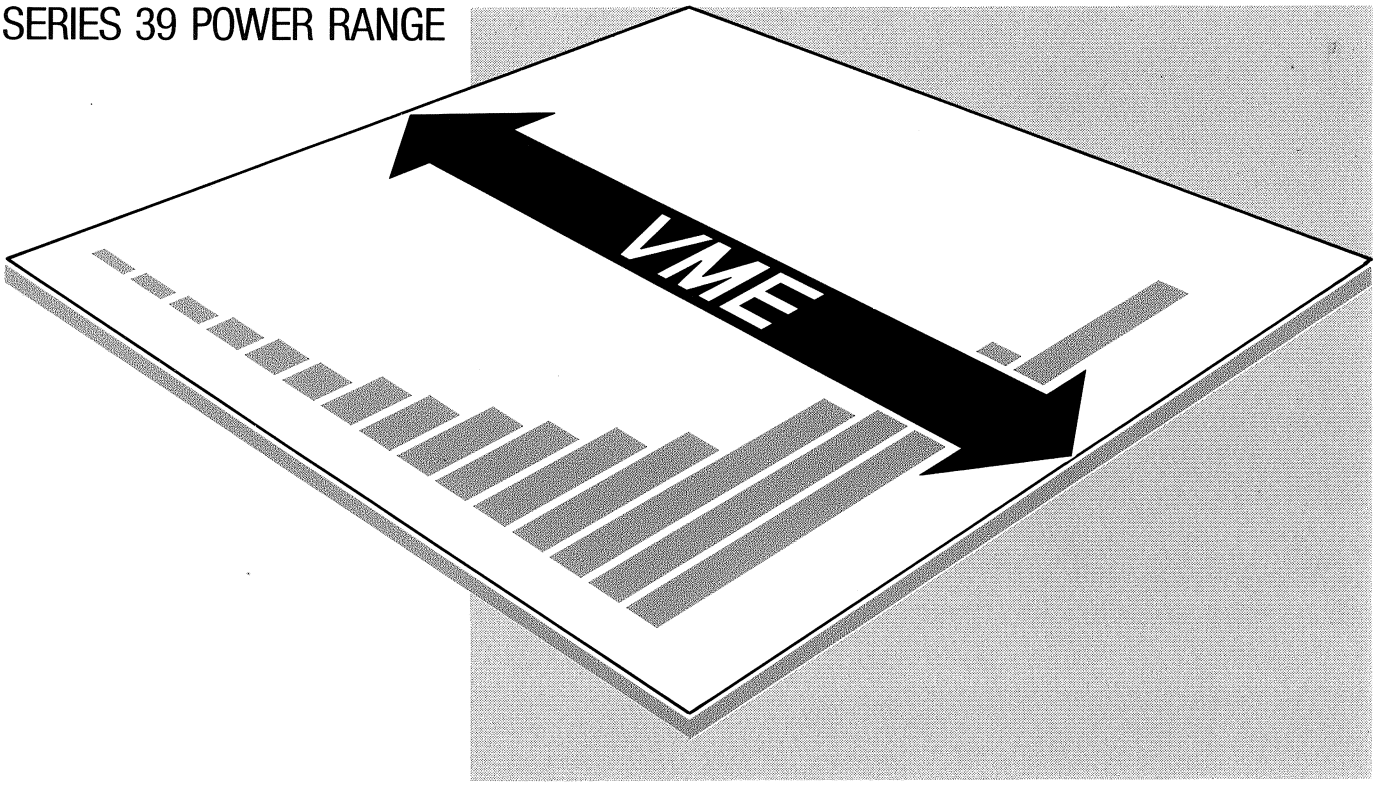
Launched in 1985, ICL Series 39 combines this innovative design with highly advanced technology. ICL's outstanding design skills, in collaboration with the Fujitsu Corporation of Japan, have produced a range of highly competitive, powerful and compact systems.

The Series 39 range allows the building of systems which are differentiated by their processing power and their peripheral configurations. The various power Levels are grouped under two headings — Distributed Systems and Large Mainframes.

Series 39 Distributed Systems are designed for the office environment. They offer new standards of flexibility and growth together with reliability and ease of use. They can be delivered with the application software ready to run and these QuickStart systems provide the ideal introduction to the powerful and flexible world of Series 39.



SERIES 39 POWER RANGE



Series 39 Large Mainframes are designed for the traditional computer room. These systems extend the Series 39 performance range, and with the highly resilient 2, 3 and 4 node configurations they provide the most powerful transaction processing systems available.

Each Series 39 system is supplied with CAFS, ICL's unique information searching mechanism, enabling information held on disc files to be searched at many times the speed of conventional methods. This product is recognised internationally and won for Mainframe Systems the 1985 Queen's Award for Technological Achievement.

ICL VME is the operating system used by all Series 39 Levels. VME covers the entire spectrum from Level 15XP to the four node Level 80, representing a performance range of 1:50.

VME is used on more systems than any other mainframe operating system in the UK. It is a truly on-line, virtual environment, networking system. It has been developed for over a decade, refined, enhanced and is now fully mature. With over 3000 man years of highly skilled and creative effort, VME is well proven and has superiority over all its competitors, offering the best mixed-mode operating environment in the world.

VME provides advanced computer facilities for both computer professionals and end-users who have no specialised knowledge. It is the operating system which can be used with equal success in a small departmental application or a vast, complex, international network supporting thousands of simultaneous users, with compatibility across the entire range.

As well as having the right systems, ICL customers need a comprehensive range of services. ICL is committed to providing a total service solution. Customers can choose the service options which are most appropriate to their short and long term strategic requirements. ICL was the first mainframe service organisation to qualify for the British Standards Institution's BSI 5750, awarded to companies who meet the standards necessary to provide the highest levels of service.

People

The strength of ICL lies in the high calibre of its people, working in a dynamic and responsive organisation.

Good people must be able to feel that they are a part of the company and job satisfaction is a high priority in ICL. This comes from both financial reward and the opportunity for career progression.

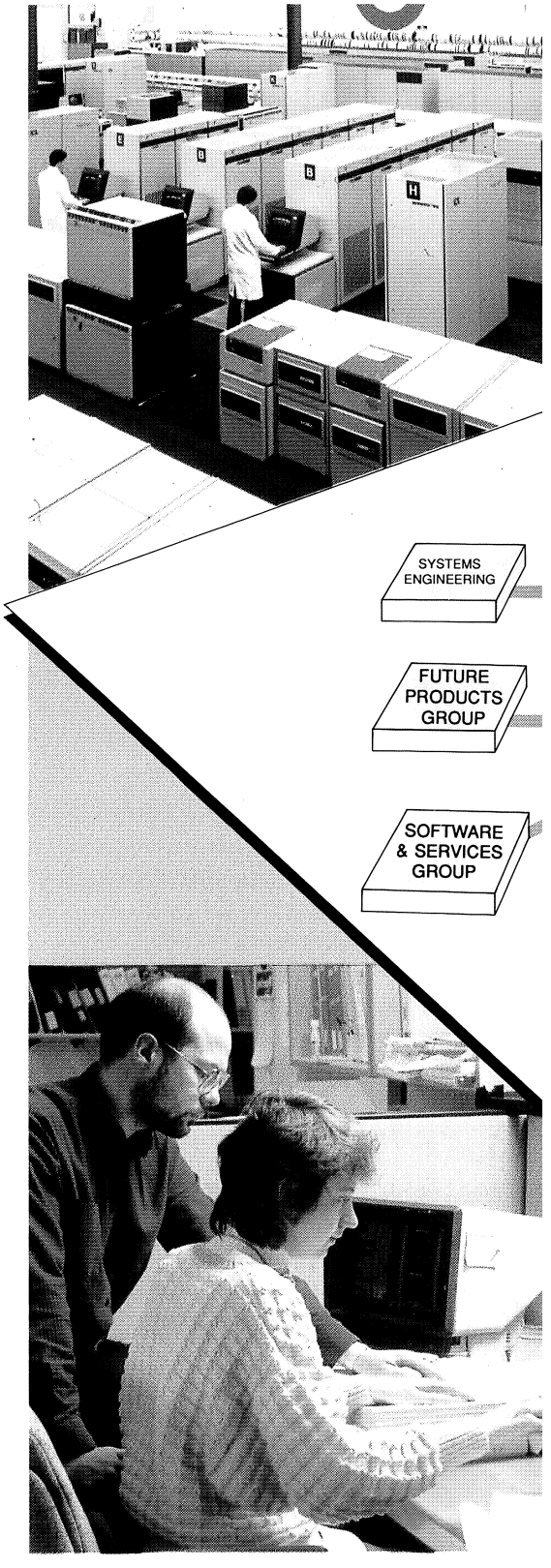
Company wide induction and technical training is given not only on joining but throughout the individual's career. ICL's commitment to developing and rewarding people is reinforced by a company wide programme placing clear requirements for this on managers.

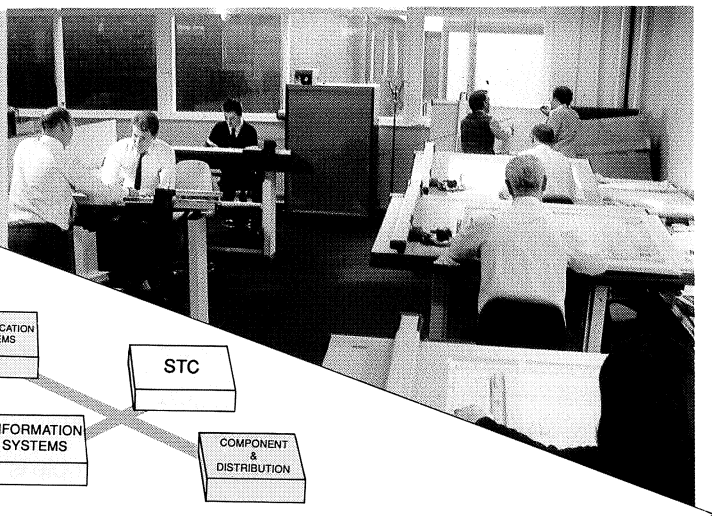
Having the best working environments, training and career development opportunities together with excellent social facilities ensure that all employees achieve their full career potential. ICL won a National Training Award in 1988 for its company wide Quality training programme.

Mainframe Systems employs people with a wide range of professional qualifications covering all aspects of the business world. The best people deliver the best results.

The Mainframe Systems Marketing unit formulates the marketing strategy for the division and provides the essential link between the needs of ICL's customers and the work of the development teams.

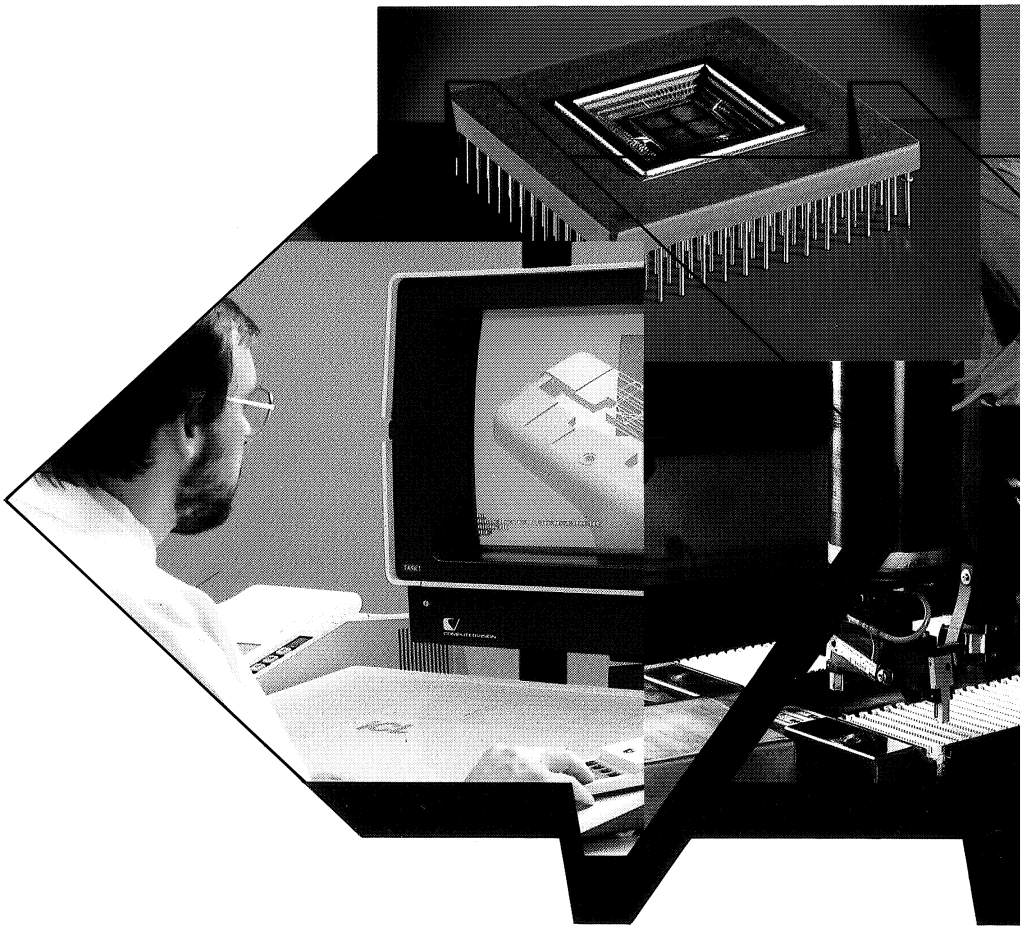
The quest for quality is fundamental to the ICL philosophy. Along with the rest of ICL, Mainframe Systems is dedicated to meeting its customers' requirements **first time, on time, every time.**





COMMUNICATION SYSTEMS
STC
UK
INFORMATION SYSTEMS
COMPONENT & DISTRIBUTION
MANUFACTURING & LOGISTICS
EUROPE
ICL
BUSINESS FINANCE
OFFICE SYSTEMS
PRODUCT OPERATIONS
INTER NATIONAL
RETAIL
PROGRAMMES
MAINFRAME SYSTEMS
NETWORK SYSTEMS
CROSS RANGE SOFTWARE & SERVICES
MARKETING
PERSONNEL
SERIES 39 PRODUCTS GROUP
SYSTEMS VALIDATION GROUP

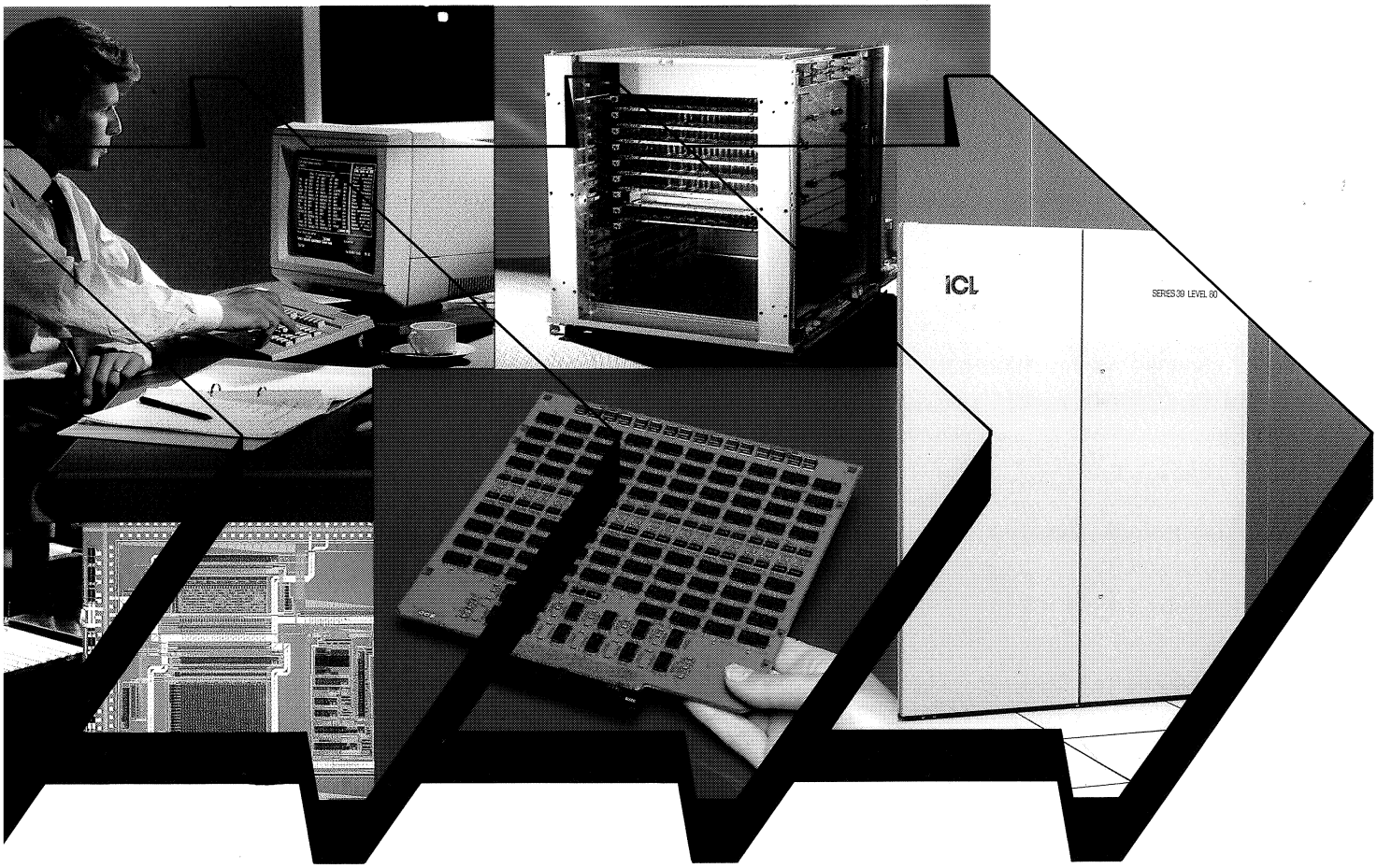
Activities



Development is carried out in five Product Groups. These Groups work on the whole range of activities, from the choice and provision of the basic technologies, right through to the rigorous testing of the finished product.

Systems Engineering is responsible for the definition of system architectures and for identifying and selecting the most appropriate technologies. It also provides design tools, including extensive computer aided design facilities.

Future Products Group designs and develops new state-of-the-art hardware systems to ensure that ICL will continue to meet the ever growing demand for more power and better price performance, through the 1990's and beyond.



Software & Services Products Group is responsible for the development, maintenance and support of VME and services facilities, including database and transaction processing management systems. This Group also undertakes the release of VME and produces customer information on all Mainframe Systems' products.

Series 39 Products Group undertakes the validation, integration and trialling of the expanding current range of hardware, software and services, and future products. This Product Group also undertakes the development of communications and peripheral systems. They are also concerned with the validation of all system support tools and special manufacturing requirements.

Systems Validation Group undertakes extensive programmes of complex system validation and collaborative activities with ICL's key customers.

All these units are supported by very experienced Personnel and Finance staff.

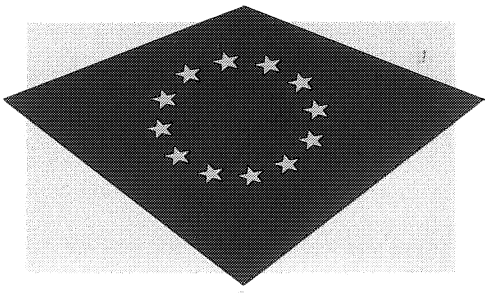
Customers

ICL operates in over 70 countries around the world. Almost half of the total output is exported.

Since ICL Series 39 was announced in 1985, over 1500 systems have been sold in places as far apart as Scandinavia, Australasia, the Far East and the Caribbean.

The advent of the single European Market in 1992 presents further opportunities and ICL will concentrate on Europe as its home market.

In the UK, ICL is the lead supplier to many sectors. Central Government use ICL mainframes for Income Tax, Social Security benefits, VAT collection, Savings Certificates, Civil Service pay and pensions and for many other applications. Similarly in local government ICL is the major supplier to over 250 authorities providing an unrivalled applications portfolio of both in-house and collaboratively produced software.



ICL has long maintained a considerable presence in the Gas, Electricity and Water industries and ICL systems currently handle over half the British Gas customer bills and the production of over 80% of water bills.

ICL has been the main supplier to key City organisations for many years. ICL systems are used to process up to 12 million inter-bank transactions daily at BACS, the world's largest automated clearing house, settle 75% of all UK credit insurance and provide real-time, multicurrency accounting to over 150 international banks.

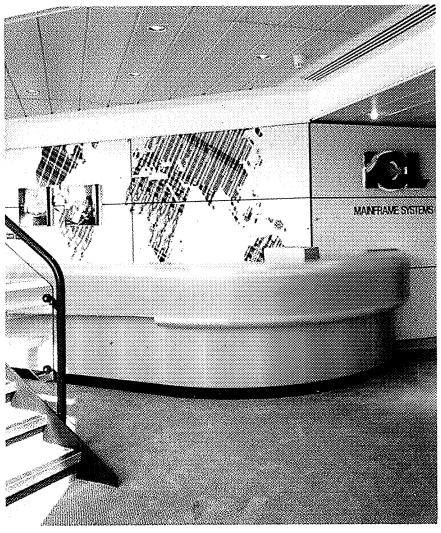
In addition to the large and famous name customer accounts, ICL also sees the small Series 39 customers as essential for future development and growth. Typically these represent small or relatively new businesses where the scope for growth is considerable as the company expands, or where there are new areas for application of computers which need to be developed and supported.

Location

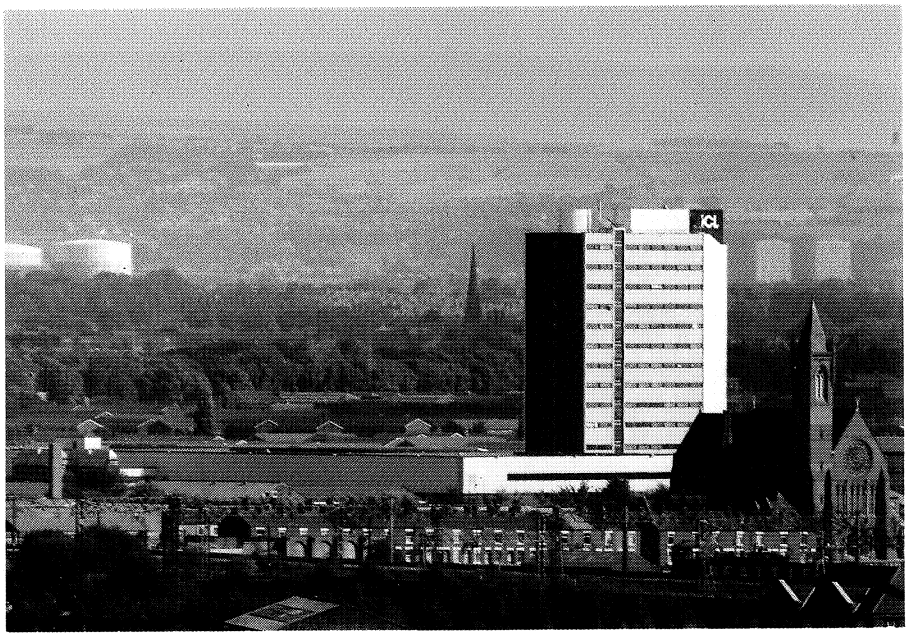
The Headquarters of ICL Mainframe Systems, is located in West Gorton on the outskirts of Manchester, and is well served by first class national and international communications links.

The site occupies some 13.5 acres and around 1300 Mainframe Systems staff are employed there. It boasts one of the largest collections of computing power under one roof in Europe. But it has not always been like this.

Manchester University was one of the handful of universities around the world to create a digital computing system in the 1940s. Subsequently, in collaboration with the Ferranti Company, they developed a series of systems which are frequently quoted as pioneers in computing — the Ferranti Mark 1 and Mark 2 and the Ferranti Pegasus. Ferranti's specialist computing division moved into an existing factory site in West Gorton, in order to manufacture the new systems.



From the late 1960s the shape of the British computer industry went through many changes. Ferranti sold their computer division to ICT in 1963, and ICT joined with English Electric Computers to form ICL in 1968.
In 1984 ICL became part of the STC group of companies.



During this period, the original buildings at West Gorton were gradually replaced at a cost of over £20M to provide one of the best development environments within the computer industry today.
These developments were concluded in 1988 with the opening of the £1M visitors centre.

ICL also operates major manufacturing facilities based in a brand new purpose-built plant at Ashton-under-Lyne about three miles away, and a major sales and customer support unit, operated from Arndale House in the city centre.

Manchester provides an excellent working and leisure environment being surrounded by some of the finest countryside in the UK. The Lake District, the Peak District, Snowdonia National Park, the Yorkshire Dales and the plains of Cheshire are all within easy reach of the city centre and its suburbs.

For further information contact:

International Computers Limited
Registered Office
ICL House, Putney
London SW15 1SW England

ICL IS A MEMBER OF THE STC PLC group

Produced and published by
ICL Mainframe Systems, Manchester, England.

MS.B207 Printed in England 09 89

AN STC COMPANY