THE 168/E AT CERN AND THE MARK 2, AN IMPROVED PROCESSOR DESIGN.

D. Lord, P. Kunz*, D. R. Botterill**, A. Edwards, A. Fucci, G. Lee, B. Martin, G. Mornacchi, P. Scharff-Hansen, M. Storr, T. Streater.

CERN, Geneva; *SLAC, Stanford; **Rutherford Laboratory, Chilton.

Abstact. In the first part of this paper the history of the 168/E work at CERN is reviewed, this is followed by a short description of the hardware and software in use and finally some examples of future applications are given. In the second part of the paper the limitations of the present 168/E design are discussed and the ways that a new design, Mark 2, could overcome them are presented.

INTRODUCTION

In any discussion on the organisation of High Energy Physics experiments, there are a number of oft recuring themes, amongst which are:

- 'how can we record the data from even more interactions per second?'
- 'how can we ensure that a higher percentage of the interactions recorded can be analysed ?'
- 'in any case how can we analyse the many thousands of tapes that we fill with data each year?'.

For a number of years now various forms of hardwired logic and special processors have been applied successfully to the first two of these themes or problems. The present conference is evidence of this work and its interest to the High Energy Physics community. The 168/E is a special, microprogrammed processor that was originally conceived by Paul Kunz as an attempt to solve the third of these problems for a particular experiment, the LASS Spectrometer at SLAC [1,2]. This the processor does by emulating almost bit for bit a large IBM 370 computer executing programs largely written in FORTRAN. As such this is an entirely generalised solution to the problem of how to do the so called, off-line data analysis for large High Energy Physics experiments at a relatively low cost.

Some time after its conception it was realised that the 168/E was not just limited to this role of off-line analysis. Because of its speed, its ability to compute more complex algorithms, its relative ease of use and low cost, it could be applied to the second of the above problems, i.e. to do an 'on-line filtering' to improve the 'quality' of the data being recorded.

This paper first of all presents a brief outline of the past work at CERN on the 168/E followed by brief descriptions of the hardware and software at CERN. The last chapter of the first part of this paper reviews some of the future applications foreseen. The second part presents the lines on which we are thinking to improve the 168/E.

PART I.

THE PAST.

The work at CERN on the 168/E started in the Spring of 1978. The SLAC prototype with single precision floating point, running test programs had been seen. As a consequence it was

decided to build a prototype according to the design we had obtained from Paul Kunz. In this way we at CERN planned to gain experience with the hardware and software of this processor and to examine its performance and ease of application.

This prototype was completed and was operational by November 1979. Since then this first system has been our test-bed used for checking the elements of the other systems that we and the members of the experiments NA3 and NA4 have constructed. By the time that this prototype was working, the EMC collaboration had become interested in the idea of using the 168/E to run their off-line analysis program, i.e. an application similar to that of Kunz at SLAC [3]. However the size of this analysis program and its data structures largely exceeds the maximum memory size of a 168/E processor. It was necessary therefore to have a mechanism for rapidly overlaying segments of program and data from memories external to the 168/E. A means of providing such a mechanism, known as the Bermuda Triangle, had been designed at SLAC [3,4] and again, to economise time and effort, we decided to construct a Bermuda Triangle, but with a number of modifications arising from the fact that we were connecting our 168/E systems to our computer centre via CERN's standard networks (CERNET and OMNET) rather than attaching directly to a channel on one of the large IBM computers, as was done at SLAC

The first Bermuda Triangle was finished by Summer 1980 and we had started to test it with its 168/E processor. In the meantime it had been decided that we should build a second system to be used by the SFM collaboration to provide a more sophisticated on-line filter. Because of the algorithms they wished to use, overlaying of program and data was going to be necessary, so that despite the very different application, the hardware and software they required was almost identical to that needed for EMC. Both of these systems were completed and under test at the application level by the end of 1980. The experiences and successes of these two collaborations' use of the 168/E are described in two papers already presented at this conference [5,6].

In parallel with this work some people from Saclay, working at CERN on the experiments NA3 and NA4, have constructed two 168/E processors. These are connected to the CAMAC systems on their data acquisition computers, via an interface module that they developed, following a suggestion of Kunz. It is their intention, when these experiments resume operation this summer, to use the processors to do on-line filtering that should reduce the the number of events that they write to tape by some 50%.

THE 168/E HARDWARE.

The 168/E hardware was designed to emulate quite efficiently a large portion of the IBM 370/168 basic instruction repertoire. It is a microcoded processor, but is unusual in that the interpretation of the IBM 370 machine instructions into 168/E microinstructions is not done by part of its hardware at program execution time. Instead it is done as an off-line process by a Translation program. This translation is done as part of a normal batch job which runs on an IBM 370. The resulting microinstructions are then loaded into one of the 168/E's memories and executed from there.

The 168/E itself consists of an integer CPU, a floating point processor, memory, and an interface (see Figure 1). For systems where overlaying is required, then a pair of Bermuda Triangles and buffer memories are added. Integer CPU

The integer CPU circuit is based on the AMD 2901, which is an LSI bit slice microprocessor chip. This handles the following types of IBM 360/370 instructions: 16 bit integer, 32 bit integer, 32 bit logical, all conditional branching, and all memory addressing. It has a 150 nsec cycle time. The throughput on FORTRAN programs has been measured to be between 1.3 to 1.8 times slower than a 370/168. The only noticeably slow instruction when compared to the 370/168 is multiplication.

Floating Point

The floating point processor consists of two circuit boards. This processor handles all IBM 370 single precision floating point instructins with exactly the same results, bit for bit, as the 370/168. But since the single precision format of IBM contains only a 24 bit mantissa, some form of extended precision is required for High Energy Physics analysis programs. It has been found experimentally for the LASS Spectrometer at SLAC that about 8 more bits are required in the mantissa to do the calculation with sufficient precision. On

the IBM 370, one declares the important variables double precision which adds 32 additional bits to the mantissa. On the 168/E, a compromise was made between true emulation and circuit cost and complexity. So the floating point processor has pseudo-double precision instructions which add 16 bits to the mantissa. Thus the processor can do either 32 bit of 48 bit floating point arithmetic. The cycle time of the processor varies between 100 and 200 nsec depending on the type of instruction and the phase of its execution. It operates with an internal Programmable Read-Only-Memory (PROM) to control the detailed steps in the execution of a floating point instruction. The performance of the Floating Point processor is about a factor of two slower than the 370/168. Again, only multiplication is a noticeably slow instruction when compared to the 370/168.

Memory

The memory for the 168/E is in two parts, one for the program and the other for the data. Both are based on the Intel 2147 memory I.C. This circuit contains 4,096 words of 1 bit each with a 55 nsec access and cycle time. It has a unique feature in that when the memory is not being addressed, it powers down to 1/5 of its normal operating current. Thus a processor with 8 memory boards draws only as much power as one memory board plus 7/5 of one memory board. The cost of these memory circuits (there are some 450 of them in a 168/E) is a dominant factor in the cost of the processor. The current price in Europe is about $10 each in large quantities. Each memory board is divided into two parts, one containing 32 memory circuits for data and the other containing 24 memory circuits for program. That is, one memory board contains exactly 16 Kbytes of data and 4096 microinstructions which is equivalent to about 8 Kbytes of IBM object code, or about 500 lines of FORTRAN.

Interface

The 168/E is not really capable, as currently designed, of doing any input or output operations, as it has no mechanism for handling interrupts. It can only address its memories. The 168/E therefore has an interface board via which a real computer, acting as its controller and Input/Output system, is able to load the memories with program and data and to read the processed results. The interface is very simple . Either the processor or the interface controls the memory and never both simultaneously. The interface control logic can shut off the processor so that it releases the memory busses. Then the interface can take the busses, read or write to the memory, and start the processor. The interface between this board and, for example, the control computer appears as two separate FASTBUS-like ports (one for data and one for program).

Thus, before a 168/E can be attached, for example, to a PDP-11 an interface between this FASTBUS variant and its UNIBUS is required. Several alternatives for this interface exist i.e. the Bermuda Triangle and CAMFAST, both described below, and a simple one directly to the UNIBUS.

Bermuda Triangle System

As already mentioned, a typical analysis program and its associated data structures are too large to fit in the memories of the processor itself. The Bermuda Triangle system is a means of overcoming this difficulty by providing a mechanism for doing fast overlays, whilst at the same time providing a way of running several 168/E processors together (see Figure 2). In this system there are two Triangle boards, each of which is a three way interface between I/O ports to a large buffer memory, a PDP-11 UNIBUS and a FASTBUS-like bus to the 168/E processors. Data may be transfered bi-directionally between any two ports. One is used for program memory overlays and one for the data memory overlays. Overlays to both memories can done simultaneously.

The first port of the Bermuda Triangle is to the buffer memories. The program buffer memory, with up to 128 Kwords of 24 bits, is large enough to hold a program of some 20K lines of FORTRAN. This is enough for quite a large program, that of EMC only needs half of this. The data buffer memory, with up to 128 Kwords of 32 bits (512 Kbytes), is large enough to hold all the local variables and constants and copies of the constant COMMON blocks. The data buffer memory also buffers events on input and results on output. The memory used is slower but much less expensive than the memories in the 168/E. These buffer memories are implemented with general purpose memory cards purchased from Mostek Memory Systems. The cycle time is 500 nsec with an access time of 375 nsec. We have also used the backplane and chassis that Mostek provides for PDP-11/70 add-on memory.

The second port of the Triangle is the bus to the processors. It is a 50 line flat cable with TTL Tri-state drivers and receivers. The transfer uses a protocol which is essentially a primitive version of the FASTBUS protocol. A 24 bit address field and 32 bit data field are used. They are time multiplexed on a set of 32 bus lines. The four most significant bits of the address field are decoded to select one processor with the remaining bits selecting the internal addresses of the processor's memory. Thus, the bus allows direct access to any location within any processor. The rate of transfer on this bus is one word in 700 nsec. Thus the transfer rate on the data side is nearly 6 Mbytes per second and on the program side it is equivalent to nearly 3 Mbytes per second of IBM object code. Transfers between the buffer memories and the 168/E memories are always in block mode under control of the Triangles themselves.

The third port of the Triangle is attached to an interface on the controlling PDP-11. It is in this port that the CERN and SLAC Triangle designs differ. A DMA control is incorporated in this interface. From the point of view of the PDP-11, the Bermuda Triangle system appears as a special peripheral with seven control registers and a data register. Appropriate loading of these control registers will initiate transfers backwards and forwards across any of the Triangle's sides. Transfers between the PDP-11 and the buffer memories can either be under program or DMA control. The Triangles also provide packing and unpacking between the 16 bit PDP-11 words and the 24 or 32 bit words of the buffer or 168/E memories. Besides the normal end of transfer interrupts, the PDP-11 interface also contains a card that receives interrupts from individual 168/Es to announce that they have stopped processing.

CAMFAST module.

In order that 168/Es could be attached to NA4 (using a ND-100 as data acquisition computer) and NA3 (using a PDP-11/45), people from Saclay developed a CAMAC to '168/E-FASTBUS' interface module. This module is functionally similar to the 'SNOOP' module developed at SLAC [7].

With this module data can be transferred between the computer hosting the 168/E and the 168/E's memories in program or block transfer mode. This allows loading/reading of the 168/E's memories, control/status register and Program Counter. The module has an external input through which it can receive a Halt interrupt from the processor. Upon receipt of this interrupt a LAM is generated which can then be treated by the host computer.

The module may be switched into a SPY-mode, whence it can spy on Read or Write functions on the CAMAC dataway. In this way all or part of the data being read by the controller can simultaneously be entered into the data memory of a 168/E.

An internal status register is provided which can be interrogated to verify that a given function sent to the module has been correctly received and interpreted. A complete list of the functions to which this CAMFAST module will respond is given in Table 1.

Construction Techniques.

The first prototype versions of all of the boards have been done using wire-wrapping for the interconnections on general boards having the same dimensions as the Hex-boards used in DEC PDP-11 computers. As many memory cards are used and as they are quite simple, a 4-layer printed circuit version was produced some time ago and has proved very satisfactory.

The wire-wrap boards for the Integer and Floating Point processors have problems with noise and cross-talk that have only been cured by changing a significant number of 'long run' connections into twisted-pair connections. SLAC has been developing printed circuit versions of these boards, however these have to have eight layers so this has not been an easy task. We elected to use a different technique called 'multi-wire'. This has a number of advantages. The wire lists used to control our wire-wrap machine can be taken and used directly to define the multi-wire connections once the wire-wrap prototype has been debugged. This conversion from wire-wrap to multi-wire can therefore be done with the introduction of no further errors. Another important advantage is that this type of board can have very complete ground and power planes, hence noise and cross-talk problems are minimised. Finally the same density of circuits can be maintained on a multi-wire board as on a wire-wrapped board.

168/E SOFTWARE.

Preparation of Application Software.

Every program that is run on a 168/E is written in either FORTRAN or in Assembler language (but following certain rules, so that it will have the same form as the code produced by the FORTRAN H-Ext compiler). The Object modules produced by the compiler or assembler have to be converted into the 168/E's microcode. These converted modules must then be linked together with converted FORTRAN library modules and previously partially linked modules to form Load modules (one for program and one for data). It is these Load modules that are transfered via CERNET/OMNET to the controlling minicomputer for loading into the 168/E for execution (see Figure 3).

These two operations of translating and linking are done by two programs known as HYPERTRANS (see Figure 4) and HYPERLINK. These programs are run as steps in normal Batch jobs on the IBM computers of our Computer Centre.

HYPERLINK also has facilities that help the placement of COMMON blocks in the data memory. This is important when dealing with programs that have been split into overlays. In this case the use of the 168/E data memory has to be carefully planned. COMMON blocks that are used in several consecutive overlays must be kept in the same position in the 168/E data memory, whilst still leaving sufficient space for those COMMON blocks that have to be moved into memory with each succesive overlay.

In planning the way that the system should handle a program that is being overlayed, we followed a simple stategy that even if we have several processors in the same system, all of them will execute the same program and that each processor will take an event all the way through the overlays needed to process it. Furthermore we attempt to structure the overlays of such a program so that each one is brought in and executed only once per event.

These programs along with a number of other useful aids were given to us by SLAC and we have been able to use them with very little modification.

Control Software.

Although the two systems currently in operation at CERN are for two different environments, off-line analysis (EMC) and on-line filtering (SFM), because they both have Bermuda Triangles, PDP-11s as control computers and connections to our Computer Centre via CERNET/OMNET, the control software used in their minicomputers has a lot in common. In the case of the system for EMC, the whole operation is controlled by a JOB running in the IBM 370/168 that first of all transmits to the 168/E system the progam and data overlays of the program to be executed; it then loops on reading and transfering event data to the 168/E system and then getting results back and writing them to an output file. In the final stages of this job it recovers statistics on the run which it prints out.

The control software in the PDP-11 [8] is based on an I/O-driver (IE) for the Bermuda Triangle-168/E hardware, a Buffer Manager (BM) and three main tasks (INI168, ETRANS and EPT):

(a) IE. This is a specially written RSX-11M driver. Besides providing the basic functions of 168/E control and the transfer of words between the PDP-11, buffer memories and the 168/Es, it also accepts lists of functions that it should carry out. In this way when a 168/E requires a new overlay, this is handled within the driver itself, thus avoiding the considerable overhead of task switching under RSX-11M.

(b) BM. This driver is the manager of the event/result Buffers. It handles requests from the other tasks for access to the event/result Buffers in the data buffer memory and to change their status [9].

(c) INI168. This task initialises the 168/E system, opens a network link and establishes contact with JOB running in the IBM. It receives the program and data overlays via the link and asks for them to be written to the buffer memories. After this it initiates the setting up of a number of buffers in the data buffer memory to contain event data and results. INI168 then spawns the other main tasks and hands over control tables [10] to them; it

then goes to sleep until a task terminates at which time it tries to get in contact with another JOB in the IBM and then to repeat the above.

(d) ETRANS. This is the task that is responsible for the transfer of event data and results between JOB and the buffers in the buffer memory. It also controls the four modes of operation of the system i.e.:

- Non-pipelined. In this mode an event is analysed and its result is returned to JOB before the data of the next event is transfered.
- Filling. In this mode all of the Buffers in the data buffer memory are filled with event data and then the mode automatically changes to :
- Pipelined. This is the mode for normal processing. It ensures that the best throughput is obtained by overlapping the transfers of data and results with execution in the 168/E. When there is a Buffer containing a result, ETRANS sends it to JOB and then waits for JOB to return the data for a new event which it puts into the emptied Buffer. In this way ETRANS uses the Buffers to smooth out variations in speed of the CERNET links and 168/E processing.
- Flushing. At the end of a run, e.g. when a JOB comes to the end of its input file, there will in general be some Buffers containing results. This mode empties these Buffers by transfering their contents to JOB.

(e) EPI. There is one of these tasks for each of the 168/Es active in the system. It loops on obtaining a Buffer containing event data, passing a list of functions to the 168/E-driver (IE) to get it to do the necessary overlaying so that its 168/E will process that event. When the processing is done it requests BM to flag the Buffer as containing results.

Test Software.

The test software that we use can be divided into two broad categories. the first is a set of basic diagnostic programs that test in detail the processors of the 168/E itself and its memories. These programs, which were supplied to us by SLAC, are written in IBM Assembler language; after normal translation and linking they are loaded down into the 168/E where they run until an error is detected. After an error is found the contents of the General and Floating Point registers are dumped to the 168/E's data memory and then a basic test control system running under CATY, a simple interpretive language, [11] in the PDP-11 can read out the values that were dumped.

The second category is a set of programs developed at CERN that are run in the PDP-11, which make use of the 168/E-driver (IE) and are written in CATY. These programs check out for example: the buffer memories, the various types of transfer between the PDP-11, the buffer memories and the 168/E. They also check out the control of the 168/Es and the interrupts it generates. Because of the use of CATY this set of programs can be very readily changed to study any new problem when it arises.

We have standardized the external specifications for a set of FORTRAN-callable subroutines [12] for the 168/E, CAMFAST and Bermuda Triangle interfaces. In this way all of our CATY based test software is portable between systems using different interfaces.

EXTENSIONS TO EXISTING APPLICATIONS AND NEW ONES.

EMC. As mentioned in a paper [6] already presented to this conference, the data rates at the experiment itself are such that a total of between 4 and 6 168/E processors devoted to the analysis of EMC data, would be able to keep up with the experiment. As a first step towards this we are constructing and testing another processor. Tests have already started to check that all is well when a second processor interface is added to the EMC system's 'FASTBUS'. The control software has already been written with the handling of several processors built into it. Therefore we hope in the fairly immediate future to test this extra processor on the EMC system.

SFM. When the SFM isn't running it would be very convenient to be able to 'play-back' the data tapes on the local data acquisition computer's tape units and then to use the 168/E in effectively an off-line mode to carry out the full analysis of the data. Work has been going on for some time now to transfer the SFM analysis programs to the 168/E.

The most serious problem in doing this, is the need to have in the data memory the very large magnetic field map (some 200 Kbytes) of SFM. With the present data memory maximum this is not practical. So we have designed a small change to the Integer Processor that doubles the amount of data memory that can be addressed. To go with this a new double capacity memory board has been designed and tested. We expect to be able to make a machine available with this larger memory in the near future.

UA-1. The use of several 168/Es to provide a sophisticated last stage trigger for this important $p\bar{p}$ experiment is under serious consideration at present. Specific examples of the on-line uses it would have would be a refined calorimeter trigger and a trigger on the momentum of charged tracks found locally in the Central Detector. These and other triggers would certainly be of great help whilst the luminosity remains low.

The data will flow from the UA-1 detectors via five parallel Remus Branches to the Data Acquisition NORD-100 computer. To gain the maximum benefit from the use of a 168/E it should not slow down the rate at which data can be taken from the detectors of UA-1. To do this it is necessary that two 168/Es be connected to the branch carrying the data on which the trigger is to be based (ultimately triggers may be used that are based on several branches). Such an arrangement gives a type of double-buffering so that the data of an event can be read into the data memory of one 168/E and processing started without delay, in parallel with the second 168/E completing the processing of the previous event and, if it accepts that event, causing it to be read down to the NORD-100 or to Videotape. Figure 5 shows one way that this could be achieved using the CAMFAST module to interface to the 168/E. A faster version of the 168/E would certainly be of interest in this application.

CONCLUSIONS

CERN has now two operational 168/E systems that have shown that they can be effectively used for their planned applications. CERN is now in a position where it can relatively easily construct further processors to extend these applications and to equip other experiments.

PART II.

MARK 2.

The current processor is adequate for many experiments. If needed, overlaying with the Bermuda Triangle for off-line applications is not a big burden. In the case of the LASS spectrometer, for example, it accounts for less than 4% of the execution time. The effort to organize an existing application program into overlays is not unreasonable, approximately two months in the case of EMC. However future applications already present difficulties due to, for example, very large event sizes and large field maps that cannot be dealt with efficiently by using overlays. Futhermore when one considers the problems involved in manufacturing and supporting a significant number of 168/E systems, then the Bermuda Triangle is an additional complication.

Besides producing very large amounts of data, most recent and future detectors will probably require a large amount of computer time per event because, for example, the jet structure will group many individual tracks into a small volume. One would therefore like to make a faster 168/E in order to handle a large volume of time consuming events with a reasonable number of processors. Also many of the current and future applications are for on-line triggering and/or filtering. In these cases the raw speed of the processor is important unless the whole event is buffered so that multiple processors can be used to improve the throughput. To summarise, the present design of the 168/E has a number of limitations :

1. The maximum data and programm memory sizes are limited to 128 Kbytes and 32 Kwords respectively (giving rise to the need to have the Bermuda Triangle to do overlays).

2. The Floating Point processor cannot do true double precision arithmetic (i.e. 64 bit words). This considerably complicates the task of comparing results from the 168/E with those from a real IBM 370.

3. No byte addressing or byte manipulation instructions. This has caused some problems when translating the FORTLIB. This may also be a problem with code generated by the new FORTRAN 77 compiler.

4. The 168/E processor has no mechanism to give its control computer direct access to any of its internal registers. This complicates the task of testing the processor.

Memory size and Addresses

The limitation in the size of the 168/E's memories comes in part from the cost and difficulty in constructing memories several times larger and in part because the present processor design is only able to generate 17 bit data memory addresses (but the least significant bit is always zero) and 15 bit program memory addresses.

The cost of 4K Static memory circuits has now fallen to approximately $10 each in Europe compared to a price of over $40 each when we started constructing our first 168/E. 16K static memory circuits are now starting to become available with 55 nsec access time hence we can now design more compact memories, avoiding bus loading problems. At present they are expensive but we can hope that their price will fall below $40 per piece fairly soon at which time we could multiply the memory size by a factor of eight whilst only doubling the total memory cost, compared to that which it was for our first 168/E. This would then mean we could think of equiping the processor with 1 Mbytes of data and 256 Kwords of program.

The data memory addressing could be extended by arranging to have a full machine cycle to calculate the address of a data word (by adding a field of the instruction to the contents of a general register) up to full 32 bit precision. This would not change the speed of the machine for isolated load instructions. However it is possible to pipeline (i.e. overlap) this address calculation with register loading or storing for a previous instruction, in which case there would be a gain in speed. In order to realise this new method of address calculation, it would be be necessary to extend the micro-instruction word by 8 bits to 32 bits. These extra bits can be used to extend the addresses for branch instructions, i.e. this gives full 24 bit program addressing.

If these changes were incorporated into the design of a new 168/E i.e. Mark2, this would to a very large extent eliminate the need to split large programs into overlays. It should be remembered that the data memory size would no longer be limited by addressing but by cost and perhaps physical dimensions. It is possible to envisage making the processor asynchronous in its access to memory, in which case it would be possible to mix memories of different speeds, densities and costs. For example part of the data memory could be made up from cheap, high density dynamic memory circuits with a cycle times of say 500 nsec. This part could then be used for data that was infrequently accessed, e.g. field maps, with little overall loss in speed.

Floating Point

Since the existing 168/E was designed, a number of new LSI circuits have come onto the market that will greatly facilitate the redesigning of the Floating Point processor. In redesigning this part of the 168/E, as the integer processor will be 1.33 faster, it seems desirable to speed up these operations as well, in particular that of multiplication.

Using the new 8x8 and 24x24 multiplier circuits it will be possible to design a floating point multiply unit that is simple, fast and capable of full 64 bit arithmetic. However it is difficult to add to such a design the ability to effectively do floating point adds, substracts and divides. Under these circumstances the best approach seems to be to design a number of floating point functional units e.g. Register, Add/Substract, Multiply and Divide units. Each one can then have the simplest design and what is more, being separate units opens up further possibilities of pipelining of their use to gain even more in speed.

Pipe Line Control

One of the difficulties in designing a pipelined processor is the design of all of the interlock control that delays the execution of an instruction if a functional unit isn't ready or if one of the operands it wishes to use is still being generated by another functional unit, etc. However in the case of the 168/E, this is not necessary because the control of the pipelines can be built into HYPERTRANS. As a part of the translation process it is possible to schedule when each micro-instruction is executed on the basis of when the functional unit it needs becomes free and when the necessary operands are available. This has the added convenience that the processor can be checked out initially without pipelining, which will be much simpler, and then pipelining can be introduced progressively.

Diagnostic Capabilities

The task of testing and repairing the present 168/E processors is made difficult because the General and Floating Point registers cannot be accessed directly by the control computer, but only indirectly by first dumping them to the processor's memory. Furthermore there are other internal registers that cannot be accessed at all by the control computer. It would therefore be a very great aid if there was a special Maintenance Mode for the processor, under which it would be possible to access their contents via a number of special lines of the machine's internal bus.

CONCLUSIONS.

A new version of the 168/E designed on these lines has been estimated to be twice as fast as the present machine. It would to all intents and purposes eliminate memory size problems along with the need to have Bermuda Triangles and it should be quite easy to construct and test. Its application to a wide range of problems in the field of High Energy Physics problems should be easier and more flexible than for the present processor. It therefore seems quite likely that we in collaboration with SLAC, will construct such a machine in the next one to two years.

AKNOWLEDGEMENTS.

We would like thank: E. Gabathuler, V. Soergel and P. Zanella for the interest and support they have given to this project. We would paricularly like to mention amongst the many people who have given their help and enthusiasm: R. Dobinson, P-G. Innocenti, Ph. Gavillet, K. Gregson, B. Hinton, C. Switalski, R. Matthews, J-P. Porte, R. Gokieli, C. Bertuzzi, B. Rothan, M. Mur and P. Siegrist.

In the framework of our discussions on the application of the 168/E to UA-1, we would like to thank C. Rubbia, A. Norton and S. Cittolin.

Finally we would like to express our appreciation of the invaluable help we have had from many people at SLAC and the Weizman Institute, in particular: H. Brafman, R. Fall, M. Gravina, L. Lorenson and A. Oxoby.

REFERENCES.

[1] P.F. Kunz. The LASS Hardware Processor.
Nuc. Inst. Meth. 9 (1976) p.435.

[2] P.F. Kunz et. al. The LASS Hardware Processor.
Proc. 11th. Annual Microprogramming Workshop,
SIGMICRO Newsletter 9 (1978) p.25.

[3] P.F. Kunz et. al. Experience in Using the 168/E Microprocessor for Off-line Data Analysis. Nuc. Sci. Symposium. Oct. 1979.

[4] G. Oxoby. The Bermuda Triangle, subsystem of the 168/E interface scheme.
SLAC Internal Note.

[5] Ph. Gavillet et. al. On-line Use of the 168/E Emulator at the CERN ISR Split Field Magnet. Paper presented to this Conference.

[6] D.R. Botterilland A. Edwards. Experiences using the 168/E Microprocessor within the European Muon Collaboration. Paper presented to this Conference.

[7] D. Bernstein et. al. SNOOP Module CAMAC Interface to the
168/E Microprocessor.
SLAC-PUB-2413. (1979).

[8] J.G. Lee and T. Streater. Description of the
PDP-11 Control Software for the 168/E.
CERN-DD-168E/80-4,V3.4. (1981).

[9] G. Mornacchi. Buffer Manager Interface.
CERN-DD-168E/80-15,V1.1. (1981).

[10] J.G. Lee. Table Definitions for the 168/E Production Software.
CERN-DD-168E/80-11,V1.9. (1980).

[11] Language Definitions of CATY1.
UKCA/CAT1/01. UK. CAMAC Association, London EC1. (1977).

[12] D.R. Botterill. FORTRAN callable Interface Rourtines to the 168/E Driver.
CERN-DD-168E/80-8,V2.0. (1980).

ACTION	FUNCTION
Initialise the module	F9 A0
Read the module status word	F1 A14
Load the local interface register with a Fastbus address prior to read. The address is loaded by performing this function twice, the first loads the 16 high bits, the second the 16 low bits and intialises the Fastbus transfers	F17 A0
Read the contents of a Fastbus address. The function is performed twice, the first time to read the 16 high bits, the second to read the 16 low bits and terminate the transfer	F0 A0
Load the local interface register with a Fastbus address prior to write. Used twice as above.	F17 A1
Write to a Fastbus address. Used twice as above.	F16 A0
Set block transfer mode (before a Fastbus cycle). The twos' complement of the word count (16 bits) must be placed on the CAMAC WRITE lines.	F18 A0
Set SPY mode on the CAMAC READ lines.	F26 A0
Set SPY mode on the CAMAC WRITE lines.	F26 A1
Check the validity of the Fastbus dialogue eg read the 2 half words on the Fastbus with 2 operations of this function after successive data writes.	F0 A1

TABLE 1. - FUNCTIONS OF THE CAMFAST MODULE.

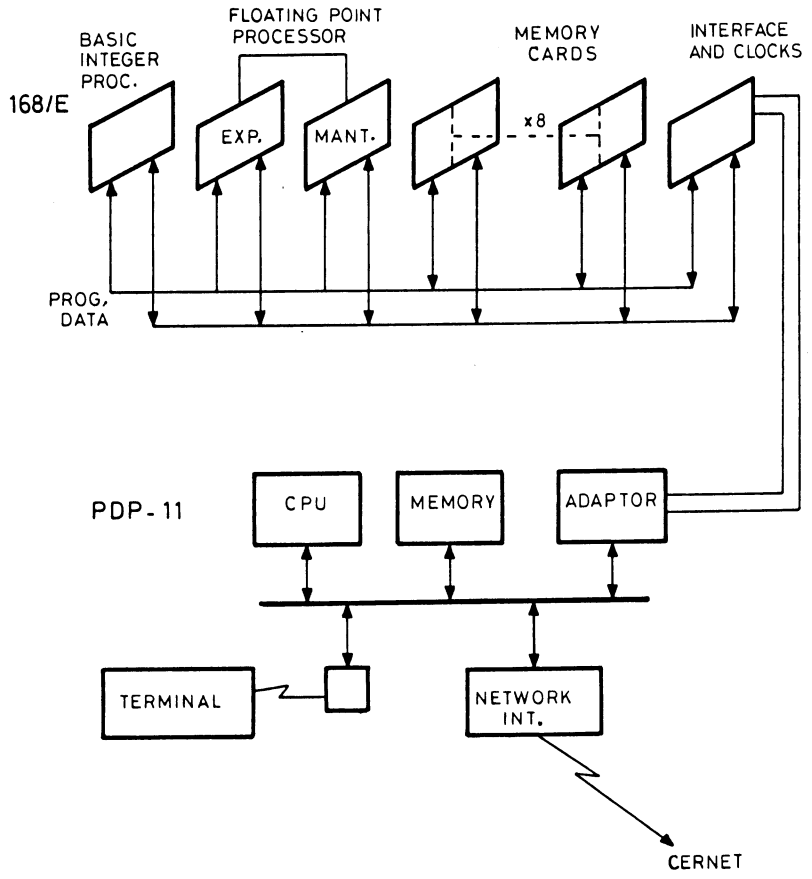

Fig. 1 Simple 168/E configuration

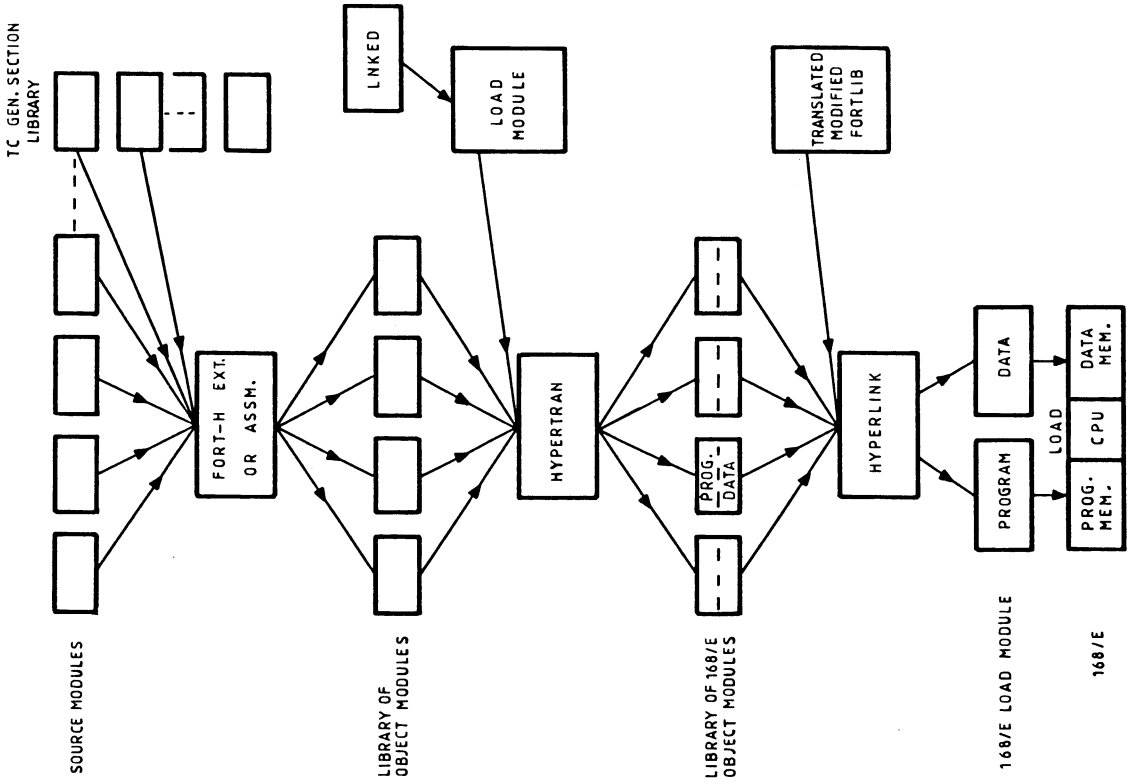

Fig. 3 Preparation of application software

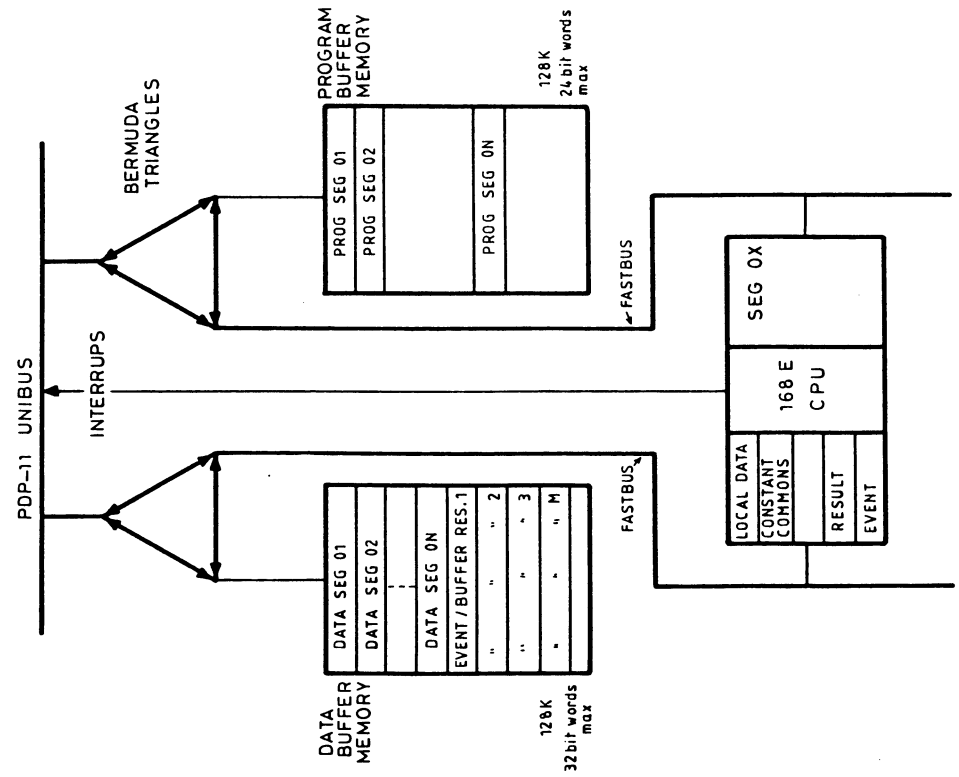

Fig. 2 Bermuda triangle system

L	0,80(0,13)	DM2	000078	000050	L	0,80		CH						
						LF	000011	3680050			0	050		
						MPS	000012	08000DF	0	0	0	337		Y<B<D
ST	0,96(0,13)	DM2	000088	000060	ST	0,96		CH						
							000013	3600060			0	060		
						WRYF	000014	0B80073	0	0	0	163		Y<B
L	0,84(0,13)	DM2	00007C	000054	L	0,84		CH						
						LF	000015	3680054			0	054		
						MPS	000016	08000DF	0	0	0	337		Y<B<D
ST	0,92(0,13)	DM2	000084	00005C	ST	0,92		CH						
							000017	360005C			0	05C		
						WRYF	000018	0B80073	0	0	0	163		Y<B
L	6,92(0,13)	DM2	000084	00005C	L	6,92		CH						
						LF	000019	368005C			0	05C		
						MPS	00001A	080C0DF	0	6	0	337		Y<B<D
SLL	6,2													
						PDC	00001B	050000E				00E		
						SML	00001C	0E4C1DB	0	6	0	733		B<*2<Y<B
L	7,120(0,13)	DM2	0000A0	000078	L	7,120		CH						
						LF	00001D	3680078			0	078		
						MPS	00001E	080E0DF	0	7	0	337		Y<B<D
L	0,0(6,7)													
						MPS	00001F	080EC41	0	7	6	101		Y<A+B
						LFX	000020	0780000			0	000		
						MPS	000021	08000DF	0	0	0	337		Y<B<D
C	0,100(0,13)	PM	00008C	000064	C	0,100		CH						
						LF	000022	3680064			0	064		
						MPSS	000023	092004D	1	0	0	115		Y<A-D
						C	000024	1270026			E	0026		
						MPSS	000025	0900043	0	0	0	103		Y<B
L	5,140(0,13)	DM2	0000B4	00008C	L	5,140		CH						
						LF	000026	368008C			0	08C		
						MPS	000027	080A0DF	0	5	0	337		Y<B<D
BCR	3,5													
						MPS	000028	080A043	0	5	0	103		Y<B
							000029	0298000			3	0000		
L	0,92(0,13)	DM2	000084	00005C	L	0,92		CH						
						LF	00002A	368005C			0	05C		
						MPS	00002B	08000DF	0	0	0	337		Y<B<D
A	0,84(0,13)	DM2	00007C	000054	A	0,84		CH						
						LF	00002C	3680054			0	054		
						MPSS	00002D	09000C5	0	0	0	305		Y<B<A+D
ST	0,92(0,13)	DM2	000084	00005C	ST	0,92		CH						
							00002E	360005C			0	05C		
						WRYF	00002F	0B80073	0	0	0	163		Y<B
C	0,88(0,13)	PM	000080	000058	C	0,88		CH						
						LF	000030	3680058			0	058		
						MPSS	000031	092004D	1	0	0	115		Y<A-D
						C	000032	1270034			E	0034		
						MPSS	000033	0900043	0	0	0	103		Y<B
L	5,128(0,13)	DM2	0000A8	000080	L	5,128		CH						
						LF	000034	3680080			0	080		
						MPS	000035	080A0DF	0	5	0	337		Y<B<D
BCR	12,5													
						MPS	000036	080A043	0	5	0	103		Y<B
							000037	02E0000			C	0000		
L	5,144(0,13)	DM2	0000B8	000090	L	5,144		CH						
						LF	000038	3680090			0	090		
						MPS	000039	080A0DF	0	5	0	337		Y<B<D
BCR	15,5													
						MPS	00003A	080A043	0	5	0	103		Y<B
							00003B	02F8000			F	0000		
L	0,92(0,13)	DM2	000084	00005C	L	0,92		CH						
						LF	00003C	368005C			0	05C		
						MPS	00003D	08000DF	0	0	0	337		Y<B<D
ST	0,96(0,13)	DM2	000088	000060	ST	0,96		CH						
							00003E	3600060			0	060		
						WRYF	00003F	0B80073	0	0	0	163		Y<B
SR	15,15													
						MPSS	000040	093FEC9	1	F	F	311		Y<B<B-A
L	14,0(0,13)	DM2	000028	000000	L	14,0		CH						
						LF	000041	3680000			0	000		
						MPS	000042	081C0DF	0	E	0	337		Y<B<D
BCR	15,14													
						MPS	000043	081C043	0	E	0	103		Y<B
							000044	02F8000			F	0000		

Original IBM Code | 168/E Mnemonic Memory Address | 168/E Program Instruction

Fig. 4 An example of translation into 168/E microcode

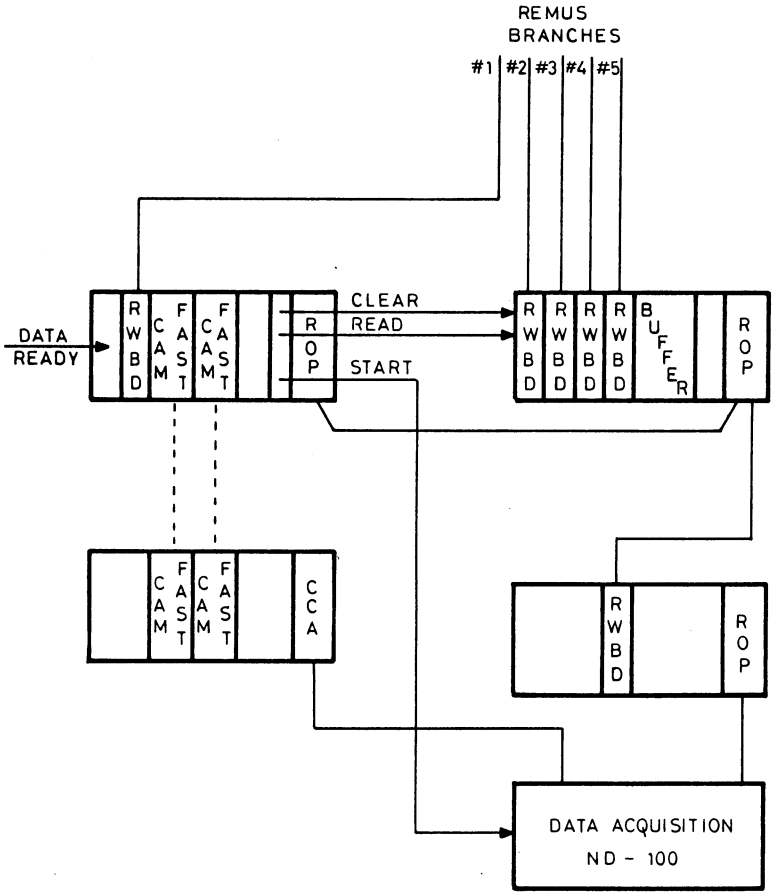

Fig. 5 Possible layout to use a double 168/E system on one branch of the UA-1 read-out