SLAC-PUB-3332
CERN/DD/84/4
April 1984
(E/I)

THE 3081/$_E$ PROCESSOR*

P. F. KUNZ, M. GRAVINA, G. OXOBY, P. RANKIN, AND Q. TRANG
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94305

and

P. M. FERRAN, A. FUCCI, R. HINTON, D. JACOBS,
B. MARTIN, H. MASUCH, AND K.M. STORR
CERN
1211 Geneva 23, Switzerland

1. Introduction

Since the introduction of the 168/$_E$,[1] emulating processors have been used over a wide range of applications[2] including both offline event reconstruction and Monte Carlo applications, and online triggering and filtering.

This paper will describe a second generation processor, the 3081/$_E$. This new processor not only has much more memory space, incorporates many more IBM instructions, and has full double precision floating point arithmetic, but it also has faster execution times and is much simpler to build, debug, and maintain.

Nonetheless, with the 168/$_E$, valuable experience has been gained on how to make efficient use of this kind of processor which, unlike computers or commercial microprocessors, does not run an operating system nor have a direct connection to I/O devices. The 3081/$_E$ takes advantage of this experience by maintaining the same style of flexible but simple interface as the 168/$_E$. This paper will also describe how such processors have been and will be used.

* Work supported by the Department of Energy, contract DE-AC03-76SF00515

Invited paper presented at the Symposium on Recent Developments in Computing, Processor, and Software Research for High-Energy Physics, Guanajuato, Mexico, May 8-11, 1984

2. The Processor

The architecture of the 3081/E is shown in Fig. 1. The details of the processor have been given elsewhere,[3] so only a brief summary will be given here. The processor has a modular structure. There are four execution units interfaced to two 64 bit wide busses, called the ABUS and the BBUS. The busses each carry 8 bytes of data and 1 parity bit per byte. Also interfaced to these busses are the control and register unit, data memory, and the interface. The control and register unit serves four functions: it contains the microprogram address counter, conditional branching logic, the data memory address logic, and the register files. A microinstruction can transfer two operands simultaneously on the ABUS and BBUS busses from data memory and/or registers to an execution unit. The results from an execution unit are transferred on the BBUS to a register, to memory, or along with a new operand on the ABUS to another execution unit. Instructions are fetched on a third, 32 bit wide bus, the PMD bus (not shown in Fig. 1). There is a single clock which has a cycle time of 120 nsec.

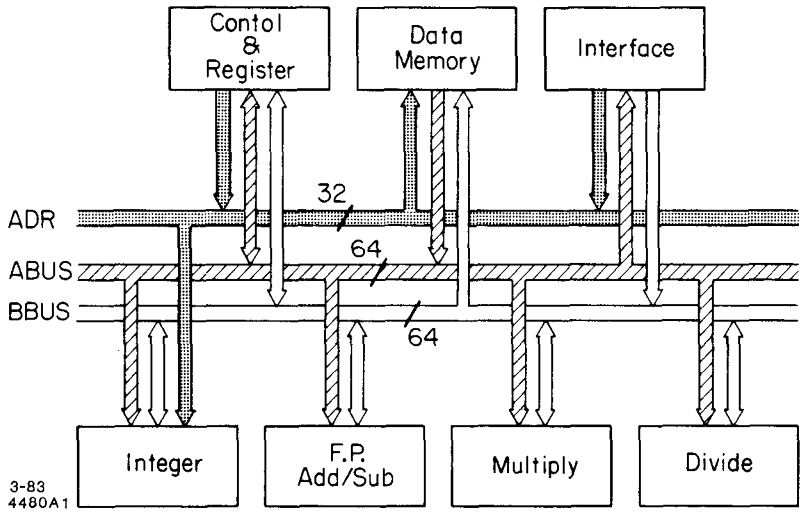

Fig. 1. Block diagram of 3081/E

An important goal of the 3081/E processor project is to produce a processor that is simple, reliable, and easy to debug and maintain. The choice of the modular architecture helped tremendously to reach these goals. The design of the processor is much simpler than the 168/E. The design is much more conservative and uses off-the-shelf multiple source TTL components. Every effort was made to reduce the man-power cost to build, debug, and maintain the processor. FORTRAN simulations have been done of each execution unit which have made a valuable contribution to the designing and in debugging. For example, the Add/Subtraction execution unit with over 200 MSI circuits, had only one design error when it was debugged, and this error was just one signal that had the opposite polarity in the hardware due to an error in the simulation. The cost of the processor, power supply, and chassis is expected to be under US$ 10,000 excluding the cost of memory.

2.1 MEMORY

Memory is one of the most important aspects of any computer or processor. In the high energy physics field, both the size of analysis programs and the quantity of data per event have grown so that the memory space needed is measured in units of Megabytes.

The memory of the 3081/E is implemented using the less dense but faster static memory circuits. Today they have 55 nsec maximum access and cycle time, come in packages of 16 Kbits, and cost about US$ 5,000 per Megabyte. The speed of memory is important because even with the best of compilers, a processor still obtains one operand (of the two for an arithmetic instruction) from memory over 75% of the time. Thus the speed of a processor tends to be dominated by its memory access time. The fast memory and 64 bit data path to it is also the best solution for online applications which must support FASTBUS I/O rates.

A 3081/E memory board at present contains one half Megabyte of either program or data memory with byte parity. The processor can accept a maximum of fourteen memory boards or 7 Megabytes. It is expected that 64K static memory circuits will be introduced in 1984 so by 1985 they will be reasonably priced. Their use will lower the cost of the processor's memory and make it possible to have a processor with 28 Megabytes.

2.2 EXECUTION UNITS

For high energy physics code, good floating point performance is essential, especially due to the heavy use of trigonometric functions in most analysis codes for solenoidal detectors. Attempts to use commercially available microprocessors with their floating point co-processors have led to disappointingly poor performance.

The following sections give a short description of each of the execution units.

Floating point add/subtract

A REAL*4 or REAL*8 add/subtract is done in 360 nsec, including reading one operand from memory. The floating point compare instruction needs only to generate the condition codes and not a result, thus it is one cycle shorter. This execution unit is also able to do an integer to floating point conversion in 360 nsecs.

Multiply

The implementation of the multiply execution unit has been optimized for single precision execution time. INTEGER*4 and REAL*4 multiplies take 360 nsec including reading one operand from memory. Modern, multiple-sourced (thus cost competitive) 16 by 16 multiplier circuits are used. To implement double precision multiplication in the same way would take a considerable number of circuits, therefore, an iterative technique is used that is reasonably fast. The results of a REAL*8 multiply is available after only 4 internal cycles for an overall time fo 720 nsecs including reading one operand from memory.

Divide

The divide execution unit does division iteratively, 2 bits per cycle which leads to a INTEGER*4 divide in about 2 μsecs and a REAL*4 divide in about 1.5 μsecs.

Integer

All integer instructions except multiplication and division are done in the integer execution unit. This unit handles the four–byte (INTEGER*4) and two–byte (INTEGER*2) arithmetic operations, and also the instructions with one–byte operands (LOGICAL*1 and CHARACTER*n). This is especially important for implementation of the instructions required by the FORTRAN '77 compilers. Both single word (32 bit) and double word (64 bit) shifts by any number of places are done in one cycle. Shift instructions are important for online trigger applications, when packed binary information needs to be expanded to individual words.

Optional units

It is possible to add other execution units to the 3081/E busses. For example, one could add a matrix multiplier/accumulator for lattice gauge theory calculations, PROM based look up tables, or other specialized 'hardware subroutines'. For the moment such devices are beyond the scope of the 3081/E project. They are also less necessary as the processor is already inherently very fast. It will also be possible to upgrade any of the existing execution units, when sufficient technology advances warrant the change, thus achieving higher performance and/or lower cost.

2.3 Instruction Pipelining

The separation of execution units, each capable of operating on its operands internally, allows for instruction pipelining. First there is pipelining of memory address calculation on the control and register board. Secondly, the Add/Subtract and Multiply execution units are capable of pipelining internally. That is, they can accept a new operand pair every cycle, then output the results in the next two cycles. Thirdly, one cycle can send an operand pair to say the add/subtract unit, and the next cycle can send an operand pair to the multiply. Fourthly, in the same cycle an execution unit can output results and another execution unit, or memory, can accept the results, thus overlapping input and output cycles. In addition, the separation of program and data memory and the separate program data bus means that program and data memories are accessed simultaneously.

Pipelining leads to substantial performance improvements in typical high energy physics code. For example, the following line of FORTRAN code:

$$\mathrm{XC} = \mathrm{VIX} * (\mathrm{XA} - \mathrm{XZERO}) + \mathrm{VIY} * (\mathrm{YB} - \mathrm{YZERO})$$

would require 23 cycles without pipelining, but only 14 with the pipelining capabilities of the 3081/E.

2.4 The Microcode and the Translator

The processor's instruction set is not that of the IBM, but is its own microcode, which resembles that of a Reduced Instruction Set Computer (RISC).[4] One could in principal write a compiler to generate the microcode, as done with IBM's 801 project,[5] instead it is generated by a software program, called the Translator. This program reads IBM object code modules, translates them to object microcode, links them together to form an absolute load module for the processor, thus using the IBM object code as an intermediate language. The source of the IBM object code could be the output of a FORTRAN compiler from any IBM compatible vendor or that of a linkage editor on either the VM/CMS, MVS, or MVT operating systems. For all practical purposes the translator step has little impact on the user. It can be looked on as a modified compile or link step. The user will be no more concerned with the 3081/E microcode then he would be about the object code from the compiler.

The microcode requires more memory space then the object code. The expansion factor is three in the worst case of no pipelining, and 1.2 in the case of complete pipelining. Nevertheless, at least 30,000 lines of FORTRAN source code can be accommodated per Megabyte of program memory, and many more lines when pipelining is generated.

The advantage of using a translator is the elimination of the complex hardware that decodes IBM instructions into microinstructions. This hardware, called the I-unit by IBM engineers, can consume well over half the total design and debugging effort of a processor. A further advantage of using the translator with the 3081/E is that instruction pipelining will be generated with a full knowledge of the context of each instruction.

2.5 Interface

The interface to the 3081/E processor is of the same style as the 168/E's. That is, either the CPU or the interface has control of the internal busses. When the processor is not running, all of the processor's memory is directly addressable through the interface. The processor thus appears as a simple slave device on, say, a FASTBUS cable segment. The transfer rate to or from the processor could be over 32 Megabytes per second if FASTBUS cable segments were sufficiently fast or 64 Megabytes per second if a 64 bit interface bus were used. VME and CAMAC interfaces are also being considered.

There are features to make it easier to debug the processor and/or program. The interface halts the processor if there is a parity error on the ABUS, BBUS, or PMD bus. The interface also has registers to allow one to halt the processor when certain conditions arise in a way similar to the Program Event Recording (PER) registers of IBM mainframes. For example, there is a stop on a Store within an address range, a stop on modification of a certain register, *etc.* Debugging some kinds of program error may be more user friendly on the 3081/E processor than it is on a mainframe computer.

2.6 Performance

To accurately predict the execution speed of the 3081/E is rather difficult, as, in common with many processors, it will depend on program's instruction mix. The pipelining of instructions makes predictions even more difficult. However, three studies have been made to predict the upper and lower bounds of the expected performance.

The lower bound of processor performance can be estimated by assuming that instruction pipelining never occurs. With this assumption the execution time of each IBM instruction is known. Ten different event reconstruction and other programs were traced while in execution to measure the frequency of instructions executed. With these numbers, the performance of the 3081/E processor would be 0.98 to 1.01 times that of an IBM 370/168.

An upper limit is estimated by assuming that pipelining occurs to such an extent that every instruction takes effectively 1 cycle. With the same samples of code, this implies execution times 2.5 times faster than an IBM 370/168; a figure that can not be realistically expected.

A third measure was obtained by translating an inner loop of one of these programs. The loop consisted of 82 FORTRAN statements containing 32 IF statements. Since IF statements break instruction pipelining, it was important to try a loop with a typical number of them. This loop also consisted of several divides and memory references with a non-zero index register. The calculated execution time for one pass through the loop for the 3081/E is 47 μsecs, while for an IBM 370/168 the time would be 71 μsecs. Thus the processor would be 1.5 times faster for this loop.

One can conclude, therefore, that the performance of the 3081/E will be at least that of an IBM 370/168 for typical high energy physics event reconstruction code, or about four times that of the VAX 11/780, and up to 50% faster under the condition that most of the execution time is spent in floating point loops. The performance of the 3081/E is comparable with a well known array processor. The FPS–164[6] has a theoretical maximum execution speed of twelve MFLOPS, while the 3081/E theoretical maximum is 8.3 MFLOPS. In practice,[7] Lattice gauge programs, implemented in microcode of the array processor, achieve about six MFLOPS, while examples of that same code, implemented in FORTRAN, would achieve four MFLOPS on the 3081/E.

3. Use of Processor

In the high energy physics environment, the use of computing resources could be put into two broad categories. The first consists of the thousands of short jobs to write and debug analysis programs, do alignment and calibrations, do physics analysis on processed events, *etc.* This category includes editing, compiling, generation of load modules, using interactive symbolic debuggers, *etc.* The second category is the long production jobs on raw data or for generation of Monte Carlo events, or in the online environment running filtering programs or analysis programs. Usually there are adequate computing resources for the first category and the limits on productivity are set by user friendliness of the operating system, response time to small needs of CPU time, the fast access to disk files, printers, graphic devices, and the memory paging of the computer. For the second category, the limits on the number of events that can be processed or Monte Carlo generated are set by the available CPU power. It is this category of processing where the inexpensive powerful emulating processors can play an important role.

As the $3081/E$ is a processor and not a computer, it, like other processors, requires support from a host computer to handle input and output operations to physical I/O devices. When multiple processors are to be used (as is frequently the case since one processor is only a fraction of a mainframe computer), this I/O support must be carefully designed for performance.[8] A multi-processor system consisting of five $3081/E$ processors, for example, will have the CPU power of a 3081K, and its I/O support system must be able to supply the data bandwidth to keep the processors busy. In practice, this means tape drives, disks, and channel rates comparable to those found on mainframe or supermini computers.

Much experience has been gained on multiple processor systems with the $168/E$ in both the offline and online environments. The planned uses of the $3081/E$ will build on this experience by preserving the same style of interface as the $168/E$ which worked well and making a few improvements in areas that only became apparent after much $168/E$ experience. The remainder of this section describes how $168/E$'s have been used and thus how we expect the $3081/E$ will be used. The interface of the $3081/E$ is designed for both the online and offline multi-processor environment. The offline environment will be discussed first as it is easier to understand.

3.1 Model of Offline Event Analysis

Consider the following model of how an event analysis program is structured. The typical program has the following steps:

1. Initialize. Initialization starts with the loading of BLOCK DATA statements into memory and continues with reading constants from disk and perhaps calling some subroutines to calculate fixed arrays that will be used in event processing.
2. Read Event. An event is read from a mass storage medium, usually tape. Checks are made to see that the record is an event record and not some other type of record on the tape.
3. Process Event. Event processing involves unpacking the raw data, generating coordinates, finding tracks, fitting tracks, *etc.* It is important to note that this processing uses much more memory for temporary variables than the initial size of the raw data. At the end of event processing, data is compressed into a block for writing to an output tape.

4. Write Event. The event is written to the output tape and the program loops back to read the next event.
5. Print Job Summary. When the event processing is complete, the program prints a summary of the job in the form of statistics gathered, histograms, *etc.*

Four remarks can be made about this structure. First, only the event processing step is CPU intensive. That is, even if the initialization or summary steps take a considerable amount of CPU time, they are only done once, thus don't really matter for a job that will run for many hours. Second, all the steps except the event processing step are I/O intensive. That is, the event processing step usually only has a few print statements for an occasional error message. Third, the program as shown above was written to run on a single processor. That is, it will process events on the same processor doing the I/O and the events are processed sequentially in the same order that they appear on the input tapes. Fourth, there is a large amount of temporary memory space used in the course of analyzing an event and, typically, a complex interrelation between this space and the program in various stages of processing.

It is therefore natural to move the event processing step to the processors, and leave a skeleton program on the host CPU for the other steps. For a single processor, the original program is modified by:

1. inserting I/O calls to download the processor with program and constant data after initialization is completed and before the first event is read.
2. replacing the processing step with I/O calls to send and receive event data with the processor.
3. and inserting I/O calls to receive the job summary data from the processor after the last event is written to tape and before the printing of job summary is started.

These modifications can usually be made with little difficulty by anyone with some knowledge of the program.

For a multi-processor environment, the program can be further modified so that it reads events and sends them to a processor until each processor has an event, then for each event received back from a processor the host program writes it to tape, reads another event, and sends to the next available processor. At the end of the job, the host program would just receive events and write to tape until all processors are empty. Also the job summary data would be received from each processor, and combined before the printing of job summary.

This model of using multiple processors allows a single host program to make efficient use of all the processors while requiring only a single set of input and output devices working on a single data stream. Letting each processor completely handle an event on its own, from input to output, avoids the difficulty of breaking up the program into stages with each stage being run on a different processor. It also allows multiple slave processors with a simple interface to be attached to a single bus for transferring data.

When the host computer is an IBM compatible mainframe, then there are the following additional advantages:

1. The initialization step can remain entirely on the host. After initialization is done, the labelled COMMON blocks with the initialized data can be downloaded to the same labelled COMMON blocks in the processors without translation of any kind. Thus the initialization code does not need any modification.
2. The output event data received from the processors can be written to tape directly without translation of any kind.
3. The job summary data can be received from a processor by direct copy from labelled COMMON blocks in the processor to the same labelled COMMON blocks on the host, thus the print summary routines do not need modification and can be called directly.

This model has in fact been realized in the use of 168/E processors at many laboratories and universities.[9] The reorganization of the original program has not been radical, indeed it is logical, and once done it has presented little problem even when, at a later date, major changes have been made to the code. In practice, the host computer may be attached to the processors via another computer with event buffers for further efficiency. The buffers allow event data to be unloaded from the processors as soon as it is ready, and new event data to loaded into processors immediately, thus causing minimum idle time on the processors and overlapping physical I/O with processing. At SLAC,[10,11] and CERN,[12–14] PDP–11s were used for attached 168/E processors as early as 1979. A Nord computer was used at DESY.[15] It is also possible for the tape drives to be on a superminicomputer, such has been done with attached 168/E processors at Toronto[16] and Saclay,[17] with some loss of ease due to differences in floating point formats.

3.2 Online Use

Multiple 168/E processor systems have been used in the online environment in a configuration that closely resembles that of the offline systems.[18,19] Similar online systems are being planned for SLC and LEP detectors.[20–24] In the online environment, the input data comes directly from the detector, being processed by the emulators before it reaches the data acquisition computer. The bus interface to the processor is, for example, FASTBUS. Everything else about the running of 'jobs' is virtually the same as the offline environment.

The 3081/E has many important characteristics for the online environment. Being an emulator of a mainframe, programs can quickly be moved from the offline to the online environment. It also has high I/O data capability to minimize deadtime, fast integer instructions including shifting and multiplies for unpacking data, large memory space to buffer data blocks from different parts of the detector, memory parity checking, *etc*. The separation of program and data memories helps avoid accidental overwriting of program in complex data acquisition systems. Dual port interfacing, which allows simultaneous loading of one processor and unloading of another is easily accomplished[8,18]. Part of the data acquisition system can plug into the internal busses of the processor as has been done with the $168/_E$.[25,26]

3.3 OTHER USES

It is clear that event orientated jobs fit well into the structure described above. But other types of job, such as simulations based on lattices or numerical integration, can also use such a system. Although one's first inclination is to put one processor per node in a lattice simulation, it has been pointed out by Fox[27] that one processor per node will lead to large inefficiencies in the processor communicating with nearest neighbors. At SLAC, nine $168/_E$ processors have been used by running the entire lattice on a single processor, but having different sets of parameters, such as coupling constants or lattice size, running on different processors simultaneously.[28]

Thus, for this type of job the 'event' is a set of parameters, each processor may work on a single 'event' for hours and the job summary printing is the comparison of the results with different parameters. These kinds of jobs require no hardware or software changes to a multiple processor system that can also run the event analysis jobs, thus various kinds of jobs can be submitted to the system just like one would submit jobs to a batch queue on a computer.

In some cases, some limited I/O capability is desirable, 'limited' is important because if I/O capability becomes very important one probably doesn't have a CPU bound job and such a job would run best on a real computer. I/O capability if it is physically done on a host computer and only virtually done on a processor is mostly a question of the software interfacing and not the processor hardware. For example, limited PRINT statements can be accommodated by the processor writing to a buffer in it's own memory, with the buffer only being read out at the end of processing an event as has been done at Saclay with the $168/_E$ processors. In the other extreme, a processor could run part of the operating system; such is the case with IBM's XT/370 where the 370 processor runs the CMS component of the VM/SP operating system while the 8088 processor of the IBM PC in which it is housed handles the physical I/O by emulating the I/O component of the operating system.

4. Conclusion

The $3081/_E$ project was formed to prepare a much improved IBM mainframe emulator for the future. Its design is based on a large amount of experience in using the $168/_E$ processor to increase available CPU power in both online and offline environments. The processor will be at least equal to the execution speed of a 370/168 and up to 1.5 times faster for heavy floating point code. A single processor will thus be at least four times more powerful than the VAX 11/780, and five processors on a system would equal at least the performance of the IBM 3081K. With its large memory space and simple but flexible high speed interface, the 3081/*E* is well suited for the online and offline needs of high energy physics in the future.

The project is being carried out as a collaboration between SLAC and CERN DD division. The work has been divided equally between them. Final debugging should occur at SLAC soon with processors being generally available for use by early 1985.

References

1. Paul F. Kunz, "The LASS hardware processor", Nucl. Instr. Meth. 135, 435 (1976).
2. P. F. Kunz, "Use of Emulating Processors in High Energy Physics", Phys. Scr. 23, 492 (1981).
3. P. F. Kunz *et al.*, "The 3081/*E* Processor", *Proc. of the Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padova, Italy, March 23-25, 1983.*
4. D. A. Patterson and C. H. Séquine, "RISC-1: A Reduced Instruction Set VLSI Computer", *Proc. Eighth Ann. Sym. on Computer Architecture, May, 1981.*
5. G. Radin, "The 801 Minicomputer", IBM J. Res. Develop. 27, 237 (1983).
6. Floating Point Systems, Beaverton, Oregon.
7. Ken Wilson, Private Communication.
8. A. Fucci and K. M. Storr, "Using 3081/*E* Emulators in On-Line and Off-Line Environments", *Proc. of the Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padova, Italy, March 23-25, 1983.*
9. A .W. Edwards, N. A. McCubbin and J. P. Porte, "Preparation of Off-line Programs for the Present 168/*E* and Recommendations for the Future", CERN DD/83/21, Oct 1983.
10. P. F. Kunz, Richard N. Fall, Michael F. Gravina, J. H. Halperin, L. J. Levinson, Gerard J. Oxoby and Quang H. Trang "Experience Using the 168/*E* Microprocessor for Off-line Data Analysis", IEEE Trans. *NS-27*, 582 (1980).
11. L. S. Rochester, "Microprocessors in Physics Experiments at SLAC", *Topical Conf. on Application of Microprocessors to High Energy Physics Experiments, Geneva, Switzerland, May 4-6, 1981.* CERN Microproc. 204 (1981).
12. C. Bertuzzi, D. Drijard, H. Frehse, P. Gavillet, R. Gokieli, P. G. Innocenti, R. Messerli, G. Mornacchi, A. Norton and J. P. Porte, "On-line use of the 168/*E* Emulator at the CERN ISR SFM detector", *Topical Conf. on Application of Microprocessors to High Energy Physics Experiments, Geneva, Switzerland, May 4-6, 1981.* CERN Microproc. 329 (1981).

13. D. R. Botterill and A. W. Edwards, "Experiences using the 168/$_E$ Microprocessor Within the European Muon Collaboration (EMC)", *Topical Conf. on Application of Microprocessors to High Energy Physics Experiments, Geneva, Switzerland, May 4-6, 1981.* CERN Microproc. 336 (1981).

14. D. Lord, P. Kunz, D. R. Botterill, A. Edwards, A. Fucci, G. Lee, B. Martin, G. Mornacchi, P. Scharff-Hansen, M. Storr and T. Streater "The 168/$_E$ at CERN and the MARK II: An Improved Processor Design", *Topical Conf. on Application of Microprocessors to High Energy Physics Experiments, Geneva, Switzerland, May 4-6, 1981.* CERN Microproc. 341 (1981).

15. T. Barklow, thesis, University of Wisconsin—Madison.

16. Steve Bracker, private communications.

17. Jacques Prevost, private communications.

18. J. T. Carroll, M. DeMoulin, A. Fucci, B. Martin, A. Norton, J. P. Porte and K. M. Storr, " Data Acquisition using the 168/$_E$", *Proc. of the Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padova, Italy, March 23-25, 1983.*

19. G. Arnison *et al.*, Phys. Lett. 126B, 398 (1983).

20. A. J. Lankford and T. Glanzman, " Data Acquisition and FASTBUS for the Mark II Detector", IEEE Trans. Nucl. Sci. *NS-31*, 228 (1984).

21. A. Lankford, Paper submitted to these proceeding.

22. The L3 Collaboration, "Trigger and Data Acquisition System of L3", CERN/-LEPC/84-5, LEPC/PR3/L3, January 1984.

23. ALEPH Collaboration, " Data Acquisition and Data Analysis", CERN/LEPC/-84-8, LEPC/M46, January 1984.

24. P. Gavillet, B. Heck and F. Udo, "Proposal for Triggering DELPHI", DELPHI Note 83-3 ELEC.

25. D. Bernstein, J. T. Carroll, V. H. Mitnick, L. Paffrath and D. B. Parker, "SNOOP module CAMAC Interface to the 168/$_E$ Microprocessor", IEEE Trans. Nucl. Sci. *NS-27*, 587 (1980).

26. J. T. Carroll, J. Brau, T. Maruyama, D. B. Parker, J. S. Chima, D. R. Price, P. Rankin and R. W. Hatley "On-line experience with the 168/$_E$", *Topical Conf. on Application of Microprocessors to High Energy Physics Experiments, Geneva, Switzerland, May 4-6, 1981.* CERN Microproc. 501 (1981)

27. G. Fox, " Scientific Calculations with Ensemble Computers", *Proc. of the Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padova, Italy, March 23-25, 1983.*

28. J. E. Hirsch, R. L. Sugar, D. J. Scalapino and R. Blankenbecler, "Monte Carlo Simulations of One-dimensional Fermion Systems", NSF-ITP-82-44.