SLAC-PUB-2198
September 1978
(T)



# THE LASS HARDWARE PROCESSOR*

Paul F. Kunz, Richard N. Fall, Michael P. Gravina

Stanford Linear Accelerator Center
Stanford University, Stanford, California, 94305, U.S.A.

and

Hanoch Brafman

The Weizmann Institute
Rehovot, Israel

## ABSTRACT

The problems of data analysis with hardware processors are reviewed and a description is given for a programmable processor. This processor, the 168/E, has been designed for use in the LASS multi-processor system; it has an execution speed comparable to that of the IBM 370/168 and uses the subset of IBM 370 instructions appropriate to the LASS analysis task.

## 1. INTRODUCTION.

In the interest of performing systematic studies in high energy nuclear physics, several large spectrometers have been constructed at CERN, BNL, and SLAC [1]. These spectrometers are capable of taking data at such a rate that the amount of computing time required for the data analysis is becoming a major problem. At SLAC, for example, the Large Aperture Solenoid Spectrometer (LASS) has the capability of recording events on magnetic tape at an average rate of an event every 10 milliseconds [2]. However, the mean time required for processing an event at the SLAC computer center is of the order of 100 milliseconds.** The goal of the LASS processor project is to process the events so as to cut down significantly the amount of computer time required to support a LASS experiment. With the advent of large detectors and relatively inexpensive read-out electronics, the problem of computer support for the data analysis in LASS is fast becoming a familiar one faced by many experimenters in high energy nuclear physics [3].

Section 2 of this paper discusses the criteria imposed on hardware processing in general while section 3 reviews the components available for implementation. Section 4 describes the programmable processor designed for use with LASS. Finally, section 5 is a summary.

## 2. CRITERIA.

In order to specify the criteria for the LASS hardware processor, a study was made of the data analyis task. It was quickly realized that the inherent structure of the task lent itself naturally to a multi-procedsor system since the overall task can be broken down into distinct sub-tasks such as unpacking the raw data, finding space points, finding line segments, etc. These sub-tasks generally try all combinations of a pair of coordinates in two planes and search for a match with coordinates in the remaining planes. This search leads to a nested loop structure of the sub-tasks which accounts for the considerable amount of computer time required for execution.

The question remained, however, as to what set of criteria one should use in selecting the individual processors. The criteria used for LASS, outlined in the following sections, are similar to those one might use in other projects whether they be single or multi-processor systems.

### 2.1 SPEED OF EXECUTION.

The effective execution speed of the overall system must be about an order of magnitude faster than a large scale computer such as SLAC's IBM 370/168. Such speeds are difficult to achieve since the 370/168 has a cycle time of 80 nanoseconds and can do a memory to register ADD in 4 cycles.

### 2.2 SPEED OF PROGRAMMING.

An aspect frequently underestimated in hardware processing projects is the programming time. Since the intention is to duplicate in hardware a

* Work supported by the U.S. Department of Energy.
** The SLAC computer center consists of a triplex system with two IBM 370/168's and one IBM 360/91.

(To be presented at the 11th Annual Microprogramming Workshop, Asilomar, Pacific Grove, CA., November 19-22, 1978)

complex algorithm which normally requires a considerable effort in software on a large computer, the means by which one will understand, write, debug, and support the program in a hardware processor is an important consideration.

### 2.3 FLEXIBILITY.

Program algorithms frequently change as one gains experience, encounters unforeseen problems, or changes the focus of the experiment in light of preliminary data. The programs may also change as new or modified detectors are brought into the apparatus, or different experiments are run on the same apparatus. It is important therefore that a processor's program can be easily modified.

### 2.4 RELIABILITY.

The processors and the system in which they are contained should be as simple as possible in order to achieve a high degree of reliability. A modular system would allow easy replacement of faulty modules or the introduction of upgraded ones. The modules should be made of components which can be replaced, if faulty, by parts readily available from stock.

### 2.5 SPEED OF FABRICATION.

The fabrication time, which includes the time it takes to design, build and debug the processing system, must take into account the talents of the people involved in the project. To be practical one should make maximum use of technologies which are already well known.

### 2.6 COST AND SIZE.

The cost and size are important criteria if one is going to have many parallel processors.

### 2.7 COMPATIBILITY.

One would like to have a system which has maximum compatibility with existing equipment including the format in which the data is presented by the detectors and the physical configuration of the apparatus.

## 3. REVIEW OF AVAILABLE COMPONENTS.

A frequently used approach to hardware processing is to build hardwired boxes with point-to-point logic.* It allows one to design an extremely fast processor. For example, one can do the calculation

$$XP = A\ X(i) + B\ Y(j)$$

in the time it takes to do one multiplication and one addition by building two parallel multipliers and separate memory banks for X and Y. Furthermore, 16 by 16 bit multiplication time can be reduced to under 200 nanoseconds by using specialized integrated circuits. This approach has been made considerably easier in recent years with the availability of MSI and LSI integrated circuits. But since the program is effectively contained in the point-to-point wiring, it suffers from certain severe disadvantages. For example, the writing of a program takes a considerable logic design effort and the debugging or changing of the program usually involves rewiring sections of the circuit. Consequently, these processors take a long time to build and debug. Flexibility is limited when the algorithm one would like to use has been simplified in order to be implemented in hardware and only a limited range of program changes are allowed without a major reworking of the circuitry. Also, reliability is impaired by the fact that the circuits are one of a kind and hence cannot be easily replaced and must be repaired by an expert when faulty. Cost and size of such processors may be reasonable but frequently they are not compatible with existing equipment. For the above reasons it was decided that hardwired processors were undesirable for a large spectrometer facility such as LASS.

The required fast effective execution speed can also be achieved by an array of programmable processors. There are many inexpensive programmable processors commercially available today which one might consider as elements in a multi-processor system. Since in recent years the cost of minicomputers has dropped considerably and their speed has increased, one might also consider their use in such a system.

In order to compare various processors, a study was made on the execution time of a simple DO-LOOP which frequently occurs in the data analysis task as the innermost DO-LOOP of many of the sub-tasks. Our studies show that for a space-point or line-finding subroutine, the repeated execution of this DO-LOOP can account for about half of the total execution time. The execution time for this DO-LOOP gives us a rough idea of the execution speed of various processors without running benchmark programs.

---

* For an excellent review with a large bibliography see C. Verkerk, "Special Purpose Processors", Proc. 1974 CERN School of Computing, Godöysund, Norway, August 1974.

The equivalent FORTRAN statements for the DO-LOOP studied are:

```
      DO 100 I=1,N
      IF (X(I) .LT. XP) GO TO 200
  100 CONTINUE
          .
          .
          .
  200 CONTINUE
```

In machine code the DO-LOOP consists of only four operations:

1) a COMPARE of a measured coordinate with a predicted coordinate in memory;

2) a BRANCH if the compare was low;

3) a DECrement of the coordinate index and;

4) a BRANCH to the top of the loop if one has not exhausted the coordinate list.

Table I shows the execution time of this simple loop for various programmable processors. For each processor except the right-most two, the code was optimized in assembly language with 16 bit integer data. The approximate cost of each processor, relative to the Intel 8080, is also given.

The two popular MOS micro-processors suffer in this comparison because of their 8-bit word size, thus the LSI-11 has a clear advantage over them. The execution time of a typical mini-computer is represented by the PDP-11/40 while that of an advanced mini-computer with MOS memory by the PDP-11/45. The PDP-11's are both micro-programmed processors, so they also represent roughly the kind of performance one could achieve by designing a mini-computer with an LSI bipolar micro-processor slice such as the 2901 series. The execution time on an IBM 370/168 is shown for comparison. One should bear in mind that to meet the real-time data rate of LASS one needs a system which is an order of magnitude faster than the 370/168. Thus in a system of parallel processors one would need 10 370/168's, 30 PDP-11/45's, 70 PDP-11/40's, 160 LSI-11's or 200 Intel 8080's. None of these options is within our budget and even if it were it is deemed extremely difficult to organize the interconnection of so many processors into a workable system.

TABLE I.

COMPARISON OF PROGRAMMABLE PROCESSORS

| Manufacturer / Model / Program Code | Intel 8080 | Motorola 6800 | DEC LSI-11 | DEC PDP-11/40 | DEC PDP-11/45 | IBM 370/168 | SLAC 168/E |
|---|---|---|---|---|---|---|---|
| COMpare Xi and Xp | 16.0 us | 16.0 us | 4.9 us | 2.5 us | 0.9 us | 0.32 us | 0.45 us |
| BRANCH LOW | 5.0 | 4.0 | 3.5 | 1.4 | 0.5 | 0.24 | 0.15 |
| DECrement i | 2.5 | 4.0 | 4.2 | 1.0 | 0.5 | 0.08 | 0.15 |
| BRANCH Greater | 5.0 | 4.0 | 3.5 | 1.8 | 0.9 | 0.30 | 0.15 |
| Total Time | 28.5 | 28.0 | 16.1 | 6.7 | 2.8 | 1.00 | 0.90 |
| Relative Cost | 1 | 1 | 1.2 | 10 | 30 | 3000 | 2 |

Another aspect of this comparison of processors is the difference in their instruction set. For example, the 6800 matches the performance of the 8080, in spite of its longer cycle time, because it requires only 7 instructions, rather than 9, for the DO-LOOP. The LSI-11, PDP-11's and IBM 370 require only 4 instructions. In general, an average programmer can produce faster and more efficient code with a processor that has a more flexible instruction set. The IBM 370/168 has a certain advantage of the processors considered, having 16 working registers which can be used either as accumulators or index registers.

None of the available inexpensive processors are fast enough nor do they have a sufficiently powerful instruction set. Consequently a programmable processor has been designed which would meet our needs. The remainder of this paper discusses the features of this processor.

## 4. THE LASS PROGRAMMABLE PROCESSOR: 168/E

The LASS hardware processors have been designed so that they are very fast, easily programmed, and relatively low in cost. Each processor has the execution speed comparable to an IBM 370/168 and, in order to minimize the programming task, the processors have been designed to efficiently emulate a subset of the 370 machine instructions.

## 4.1 PROCESSOR HARDWARE.

The processors, which have been given the name 168/E, are divided into four parts: a program memory 24 bits wide, a data memory 32 bits wide, an integer processing unit, and a floating-point processing unit. The separation of program and data memories, which allows simultaneous access to them, is an important feature for the speed of execution. Figure 1 shows a block diagram of the processing unit, and the following paragraphs discuss its various features.

The integer processing unit is contained on one circuit board. Its basic sections are the microprocessor slice array with its associated control logic, the branch control logic, and the data memory control logic. The most significant bits of the program memory (the control field) determine the control section within the integer processing unit which will execute the instruction, with data for the instruction in the remaining bits (the data field).

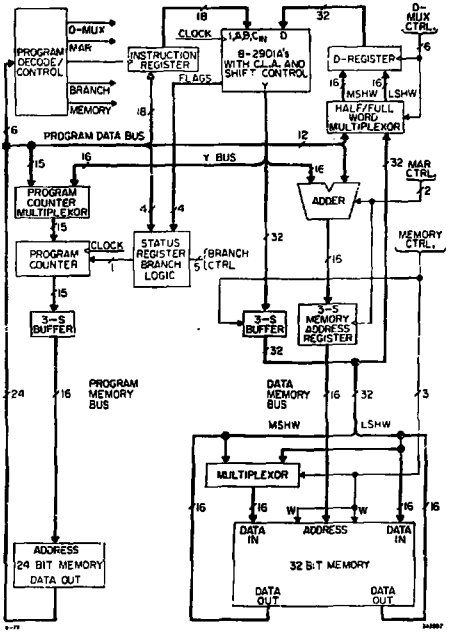


Figure 1 : Block diagram of the 168/E

The heart of the processing unit is an array of 8 bipolar LSI micro-processor 4-bit slices, the 2901A. As shown in figure 2, it comprises an 8 function Arithmetic Logic Unit; 16 adressable registers with dual port readout; an auxiliary register Q which is used for double precision shifts and multiplication; and a shifting network at the register file input ports. In addition there are status outputs to indicate CARRY or OVERFLOW conditions and ZERO or NEGATIVE results.

The micro-processor slices require 18 bits of information to execute an instruction: 3 bits to define the source operands, 3 bits to define the function, 1 for the CARRY input, 3 bits to define the destination, and 4 bits each to define the two read addresses of the register file. A register is used between the program memory and the micro-processor slices so that the instruction fetch cycle can be pipelined. The simultaneous fetch and execution feature of this processor is another

of the reasons for its high execution speed.

external to the slice array is a 15 bit binary program counter. Normally, the processor clock steps the processor sequentially through the program memory. An unconditional BRANCH instruction is executed by a parallel load to the counter from the data field of the program memory or the data output of the slices.

The status bits from the slice are not bit for bit the same as the 370/168 condition code bits, but with a few logic gates the 2901A status bits can be changed to match those of the 370/168 exactly. These modified status bits can then be loaded into the condition code register in the integer processor. A conditional BRANCH instruction is executed by placing the counter in the parallel load mode if the status bits of the slice match those in the current processor instruction.

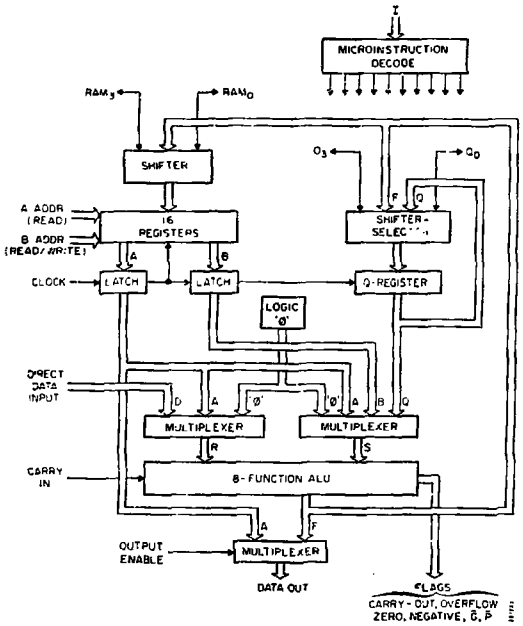


Figure 2 :
Block diagram of the 2901A

The programs for these processors can almost always be written in such a way that the BRANCH addresses are known at load time, and this address can be in the program memory data field data. Thus, most BRANCH instructions can be executed in one machine cycle.

A hardware multiplication and division algorithm has been implemented in the 168/E processors. It is done by momentarily stopping the program counter clock while cycling the slices through conditional ADD and SHIFT instructions.

In order to allow for efficient indexing of the data memory, the data memory address is formed by an ADD of bits from the data field of the program memory and from the outputs of the slices. Data may be written to the memory from the slices and similarly data may be presented to the direct inputs of the slices from the data memory or from the program memory.

All of the 168/E instruction that manipulate floating-point quantities are executed by the floating-point processing unit (not shown in

figure 1). This unit comprises a two-port register file, an ALU, a control unit, and a hardware shifting network.

When a floating-point instruction is encountered by the 168/E, the clock to the integer processor is stopped, and the floating-point processor is allowed to execute the instruction. If the instruction is one that calls for a setting of the condition codes, the appropriate information from the floating-point processor is strobed into the condition code register in the integer processing unit. If the floating-point instruction requires multiple clock cycles, the floating-point control unit stops the clock to the integer processor for as long as is necessary. In the case of floating-point instructions that require data memory accesses, the data memory address is generated by the integer processing unit, and the data is strobed into a working register in the floating-point processor.

The floating-point processor can manipulate quantities in either 32 or 48 bit precision. The 32 bit precision yields results identical to the single precision of the 370/168, while the 48 bit length is a pseudo-double precision that has been found to be sufficient for most calculations done in LASS experiments. The precision of the floating-point unit could have been extended by widening the data paths on the floating-point processing unit, but since the interconnection complexity grows rapidly above 48 bits, a compromise between cost and precision was made. This 48 bit precision mode is the only place where the 168/E does not match the 370/168 in the results it produces.

Since the floating-point processing unit is constructed separately from the integer processing unit, its inclusion in a users 168/E processor is optional.

### 4.2 PROCESSOR SOFTWARE.

Important aspects of the processor's structure will become apparent by comparison of the program code generated to perform the DO-LOOP described above. Table II shows the DO-LOOP implemented on the IBM 370/168 and the 168/E processor. The first instruction on the 370/168, cf. Table II, is a COMPARE between the contents of a memory location and register 0. The memory address is formed by the sum of register 9 (the index register), register 10 (the base register), and 12 bits from the instruction (the displacement ED). The 168/E performs the same operation in three micro-instruction cycles. In the first cycle, the slices execute an instruction which places the sum of registers 9 and 10 at its output. In the second, the displacement from the data field of the program memory is added to the outputs of the slices and loaded to the memory address register. In the same cycle, the memory is switched to the read mode and the data is strobed into a working register of the floating-point unit. In the third cycle, the comparison is made in the floating-point unit between register 0 and the working register. If the instruction had been an integer compare instead of a floating-point compare, then the integer processor would have qualified on the data memory contents. As shown in Table II, the remaining three instructions on the 370/168 can be implemented in one cycle each on the 168/E, using the integer CPU only. Thus one sees that the structure of the LASS processor allows it to emulate the IBM 370 efficiently. Emulation is possible because both the 370/168 and the 168/E have the same number of working registers, can perform the same arithmetic and logic operations, and have the same form of data memory addressing and branching.

TABLE II.

Comparison of Program Code Generated for 370/168 and 168/E

| Program Step | Code for 370/168 | Action of code for 168/E |
|---|---|---|
| COMpare Xi and Xp | LOOP CE 0,ED(9,10) | Slice: Reg. 9 + Reg. 10 -> Slice-out |
| | | Memory: ED + Slice-out -> MAR, and F.P.P: Data Memory -> Working Register |
| | | F.P.P: Reg. 0 - Working Register, and Cond. Code -> C.C. Register (COMpare) |
| BRANCH Low | BL GOTE | Branch: If less than 0, GOTE -> P.C. |
| DECrement i | SR 9,1 | Slice: Reg. 9 - Reg. 1 -> Reg. 9, and Cond. Code -> C.C. Register |
| BRANCH Greater | BNM LOOP | Branch: If greater than 0, LOOP -> P.C. |

A translator has been written which takes object code produced by the IBM Fortran H Optimizing Compiler, and converts it to relocatable program and data modules in 168/E format. An important aspect of this process is the splitting of program instructions and local data constants and variables into separate areas, ready for loading into the 168/E's separate program and data

memories. When possible, advantage is taken of the 168/E's direct program memory addressing scheme by changing IBM BRANCH instructions from their displacement plus base-register addressing format to absolute 168/E program memory addressing. Execution time saved by direct addressing applies also to the first 8K Bytes of local constants and variables in data memory, where fixed base registers may be dispensed with entirely.

After translation a linker program is used. It functions exactly like the IBM Linkage Editor, in taking all the 168/E object modules needed to compose one complete load module, and linking them together. Address constants and direct addresses are filled in or adjusted at this time. By means of control statements, the user may if he wishes position common-blocks, local data constants, and program code in specific locations, according to a predefined plan.

Not all of the 370 instruction set can be emulated by the 168/E, but all those instructions needed for track reconstruction have been emulated. In fact, the IBM FORTHAN compiler requires about the same subset of 370 instructions as those implemented in the 168/E. Those instructions not implemented deal with decimal arithmetic, character manipulation, system interrupt, and I/O. By not implementing all the instructions of the 370 one has reduced the cost and complexity of this processor while increasing its speed. The goal of this project is to build fast programmable processors for physics applications and not to build a general purpose computer with the entire instruction set of the IBM 370.

### 4.3 WHY EMULATE?

One could have built a processor with its own unique instruction set, tailored to ones needs. Instead, the 168/E is based on the architecture of the 370 for several important reasons. First of all, the writing and debugging of programs for the 168/E can be done on the 370/168 at the SLAC computer center. Once a program is running on real or simulated data, it is easily translated to the instruction set of the 168/E. Secondly, programmers who are familiar with the 370 do not require any special understanding of the 168/E processor in order to produce fast and efficient code. Even the experimenters can write programs for the 168/E because of its FORTHAN capability. In addition, the programs do not need to be debugged on a hardware box with limited I/O capabilities.

### 4.4 SPEED AND COST

Emulation of the 370 has greatly reduced the burden of programming the LASS processors. The question remaining, however, is how much emulation of the 370 has cost us in execution speed and the dollar cost of the processors. The cycle time of the 168/E is 150 nanoseconds, which is slower than the 370/168, but, as Table I shows, the executing time of the 168/E is actually competitive with the 370/168. Fast execution of the 168/E for DO-LOOP shown comes mainly from the fact that BRANCH instructions can be done in one cycle. The 370/168 operates in a multi-programming environment so that it spends many cycles calculating the absolute address of the BRANCH. The basic DO-LOOP of Table I is biased in favor of the 168/E because of the two BRANCH instructions. A better comparison was made with complete programs. Programs were written on the 370/168 using FORTRAN H OPT=2 Compiler. With real data the execution time for the program on the 168/E is no worse than a factor of 2 slower.

Most of the cost of the 168/E is in the program and data memories for the processor. The cost of 16K bytes of data memory is about $1000, while 8K bytes (equivalent) of program memory cost around $750 at current memory prices. The integer and floating-point processor cost about $500 and $1000, respectively. These prices include components, circuit boards, and power supplies, but exclude labor for assembly. Thus a complete 168/E with both processors, 96K bytes of data memory, and 32K bytes (equivalent) of program memory (roughly the largest amount needed for LASS experiments) would cost about $10,000. The important point is that the speed-cost ratio of the 168/E is sufficiently high that a multi-processor system that meets the needs of LASS is economically feasible.

## 5. SUMMARY.

The computer support for the data analysis in a high data rate physics experiment is becoming a familiar problem. The fact that the analysis task can be broken down into many simple sub-tasks has led many experimenters to thinking about using hardware processors. The processors should have, however, both high execution speed and programmability. Hardwired processors can be extremely fast but take a considerable effort to design and maintain. Commercially available programmable processors are either too slow or too costly to meet our needs, even in a multi-processor system.

The hardware processing system in LASS will be based on an array of fast programmable processors. Each processor has an execution speed comparable to an IBM 370/168 and emulates a subset of the 370 machine instructions. The programs for the processors can thus be written and debugged on an IBM 370 before translation and loading to the hardware processors. The task of programming the processors appears to be no more difficult than that of programming a large computer since programs can be written in FORTHAN. Only a small number of processors are needed to meet the goal of executing at speeds an order of magnitude greater than a large computer such as the IBM 370/168.

**Acknowledgements**

We would like to thank D.W.G.S. Leith for his support and encouragement.

**BIBLIOGRAPHY**

1) Michelini, A., Int'l. Conf. on Instrumentation for High Energy Physics, Frascati, Italy, May 1973.

2) Armstrong, G., et. al., IEEE Trans. Nucl. Sci. NS-20, No. 1, February 1973.

3) Dhawan, S., et. al., Report of the NIM/CAMAC Committee on Data Rate Requirements for Physics Applications, November 1975 (unpublished).