MICROPROCESSORS IN DETECTORS AND ANALYSIS*

Eric J. Siskind
ISABELLE Data Acquisition Group
Brookhaven National Laboratory
Upton, New York 11973

Introduction

The increasing need in high energy physics experiments for computation power for both online and offline applications, coupled with the current "microprocessor revolution," has led us to examine the use of microprocessors in various aspects of HEP computing. The following article is a brief (and admittedly somewhat biased) review of current hardware products, the costs of developing and producing hardware systems, and the costs of providing appropriate software support tools which allow one to make effective use of physicists' time, and the applicability of certain systems to the various needs of HEP computing.

What is a Microprocessor?

The term "microprocessor" is currently used to describe two distinct entities which, although somewhat related, have sufficiently different characteristics, costs, and optimum uses as to merit individual consideration. The following discussion should help elucidate the difference between a microprocessor and a microprocessor!

The first use of the term describes the processing element of a microcomputer system, as distinguished from minicomputers, midicomputers, etc. The hardware is invariably implemented in LSI or VLSI chips, with a complete processor occupying one or a few chips or substrates. Such a processor executes machine instructions which can be generated via assembly of a program written in a language which is identical for all examples of this processor (and often for an entire family of processors, e.g. LSI-11's use of the same assembly language as all PDP-11 processors), or which can be generated via compilation of programs written in FORTRAN, PASCAL, C, or other higher level languages. The execution of a single machine instruction typically requires more than one basic clock cycle of the processor. The machine instruction sets typically contain integer arithmetic, including multiplication and division, as well as logical operations (e.g. bit manipulation), and occasionally implement floating point operations with additional or even integral hardware. The execution of complicated instructions, implying multiple clock cycles per instruction, requires the use of instruction decoding and execution sequencing hardware within the microprocessor. Table I lists several important characteristics of some of the microprocessors of this type which are currently available.

The other use of the term "microprocessor" describes an engine which executes microcode,[1] as distinguished from the usual macrocode instructions. Microcode is in some sense "closer to the hardware," can have instruction formats and widths which differ from application to application using the same hardware, can describe multiple operations (in different execution units) in a single instruction, and typically specifies instructions which execute in one machine clock cycle (although the instruction may contain a field which indicates that it is to be executed multiple times in a row), with each instruction capable of less powerful manipulations than that of macrocode. As an example, it is rare machine of this type which can execute an integer multiply operation without instruction repetition, although exceptions to this rule exist (e.g. FNAL M7[2]). The hardware for such machines varies quite considerably, with the data paths often built out of MSI chips with SSI control gates in order to obtain a configuration optimized for certain types of calculations. When built in this fashion, a microprocessor may contain hundreds or even thousands of integrated circuit packages. However, LSI chips containing a "slice" several bits wide of either standardized data paths or sequencing logic are commercially available, and are typically expandable to configure machines of arbitrary word width. Such machines are referred to as "bit-slice microprocessors," and a sampling of characteristics of such devices is included in Table II. It should be noted that even with the use of such chips, the instruction format is still dictated by the hardware designer.

Table I. Characteristics of General Purpose Micro-Processors. Source - EDN. 11/11/81

Chip	Data Word (Bits)	Add.Word (Bits)	Pins (D/A)	Clock Rate (MHz)	Cost (100 lot)
8080	8	16	8/16	1-3	3.70
8085	8	16	8/16	1-5	4.40
6800	8	16	8/16	1-2	4.95-6.20
Z80	8	16	8/16	2-6	8.00-15.00
8086	16	16	16	5-10	58.50-127.40
8088	16	16	8	5	14.10
68000	32	23	16/23	6-12.5	86.00-149.00
Z8000	16/32	24	16	8	35.90
432	32	24	16	8	1470.00

Table II. Characteristics of Bit-Slice Micro-Processors. Source - EDN. 11/11/81

Chip	Width (Bits)	Family	Registers	Clock Rate (MHz)	Price (100 lot)
2901	4	STTL	16	16.67	9.95
2903	4	STTL	16	10	21.00
29203	4	STTL	16	10	?
29116	16	STTL	32	10	?
10800	4	ECL 10K	0	20	48.75
10902	8	ECL 10K	0	50	100.00
100220	8	ECL 100K	1	50	?

With the notable exception of the SLAC 168/E,[3] code for such machines is generated via meta-assembly of symbolic source code in a form which is unique to the particular processor, and is quite difficult to generate. Therefore, such machines rarely employ programs containing more than a few thousand

*Work performed under the auspices of the U.S. Department of Energy.

instructions. A typical estimate is that microcode is an order of magnitude more difficult to generate than typical machine assembly language macrocode. This results from the need to specify multiple operations in each instruction, timing problems associated with different propagation delays (i.e. varying number of transfers) associated with moving data into distinct registers, the need to consider the frequently pipelined nature of microcode execution, and the typical lack of sophisticated debugging tools. The 168/E differs from the norm in that its microcode is generated by the translation of the object modules (or load modules) produced by the compilation of FORTRAN programs on the host machine, and so frequently large volumes of microcode are produced, often requiring overlaying in the relatively large program memory.

Although such machines were first developed as a means of implementing the central processors of newer computers with more complicated instruction sets and formats without proportional increase in the amount of hardware in such a processor, they also find applications wherever special processing requirements exist. Table III includes characteristics of the central processors of various computer systems. Note that faster microprocessor clock speed does not necessarily imply faster macroinstruction execution, but that memory access times and the presence of special hardware or additional connectivity in the microprocessor data paths may have far more profound effects (e.g. the VAX-11/780 is around 5 times more powerful than a PDP-11/34, yet the latter machine has the faster microprocessor clock speed). Also, note that the fastest processing units prefer the route of more hardware rather than microcoding instructions (e.g. CDC 7600, Cray I and II). Table IV indicates the variety of microprocessors found in a typical VAX-11/780 system, while Table V gives a sampling of microprocessors which have been developed for use in high energy physics.

Table III. Micromachines in Commercial CPUs

Machine	Cycle Time (Nanoseconds)	μC Width (Bits)	μC Length (Kwords)
LSI-11/2	400	22	1
PDP-11/04	260	40	0.25
PDP-11/34	180	48	0.5
PDP-11/45	150	64	0.25
PDP-11/60	170	48	2.5
VAX-11/750	320	80	20
VAX-11/780	200	96	4+1(RAM)
IBM-3081	26	?	?
CDC-7600	25	Not Microcoded	
CRAY I	12.5	Not Microcoded	
CRAY II	4.0	Not Microcoded	

Table IV. Microcoded Processors in a VAX-11/780 System

Model	Description	ALU	μC Width (Bits)	μC Length (Kwords)
KA-780	Central Processor	74S181	96	4+1(RAM)
FP-780	Floating Point Unit	74S381	48	0.5
DW-780	Unibus Adapter	None	44	0.5
DR-780	I/O Channel	2901	40	1
RX-02	Floppy Disk	2901	16	1
DMC-11	Serial I/O Unit	74S181	16	1

Table V. Microcoded Processors Developed for HEP

Model	Description	ALU	μC Width (Bits)	μC Length (Kwords)
M7[2]	Trigger Processor	10181	64	4
BADC[4]	Digitizer Controller	2901	48	0.5
TDS[5]	Digitizer Controller	10181	32	1
168/E[3]	Mainframe Emulator	2901	24	32
VCC[6]	CAMAC Channel	2903	64	4
UPI[7]	Fastbus Channel	2901	80	2

Hardware Costs

We now turn to the costs of developing and producing microprocessor systems. There is an implicit assumption in the following discussion that the microprocessor has very little in the way of private peripherals outside of the hardware directly under its control, but instead talks to humans or media via a connection to a host computer. I note in passing that if this is not so, but instead the micro is equipped with a complete set of support peripherals, including terminal, printer, floppy disks, etc., then the cost of such a development system is currently in the range of $25K.

In developing a hardware configuration for a typical microcomputer such as an 8086 or 68000, a standard estimate might be of order a man-year of engineering plus prototyping costs, or a figure of order $100K. Considerable design savings may be effected by careful use of existing hobbyist development cards or crate/bus systems such as S-100 or Multibus (or ultimately Fastbus!). On the other hand, the engineering of a bit-slice system involves somewhat higher development costs. A crude estimate for the SLAC BADC system[8] was 3 man-years of engineering plus $100K of prototype hardware construction, for a total of $250K.

The final product board of either type of microprocessor system, containing both processor and memory, has an estimated cost in the neighborhood of $1K. The estimated processing power of a current 16 bit micro is of order a few times 10^5 instructions per second, for a cost effectiveness of order a few hundred instructions per second per dollar, while the power and cost effectiveness of bit-slice systems for those applications which can be programmed effectively on them is about an order of magnitude higher than those for the single chip processor. For comparison, note that the cost effectiveness of the best of the current midis or mainframes is only around 10 instructions per second per dollar (e.g. VAX's run around $100K for a cpu executing around one million instructions per second (MIPS) while the IBM 3081 runs a couple of megabucks for around 15 MIPS of processing power).

This set of numbers leads to two distinct conclusions. The first is that microprocessors are sufficiently more cost effective than current mainframes to merit serious study of their use in conventional compute bound HEP applications such as offline production and Monte Carlos. In addition, their cost effectiveness will shortly bring the use of high level trigger processors constructed from arrays of microprocessors programmed in FORTRAN or some equivalent language and providing instruction processing powers of order 10^5 instructions per event on data streams of order 10^3 events per second into an affordable regime. The second conclusion is that once you have engineered the hardware for a system, you should stick with the hardware until there is a clear need to engineer a new system. As an example, given an engineered 8086 in a Fastbus crate with a cost effectiveness of 200 instructions per second per dollar, one should not be tempted to develop a more cost effective piece of bit-slice hardware until that enhanced cost effectiveness will offset the $250K development, i.e. until the processing requirement exceeds 50 MIPS (50 VAX or 5 CDC 7600 equivalents!) even assuming that the new hardware has infinite cost effectiveness. In real life, there obviously may be some other overriding consideration which necessitates such a hardware development project, but it should not be cost. Similarly, one should typically not bow to the desires of your hardware engineer to play with the latest new chip which is twice as fast as the old one (and is NOT a plug-in replacement) unless he is willing to pay the development costs out of his pocket. This conclusion will gain even more strength when one adds the costs of software support to the hardware development.

Software Costs

A number of system architectures have been proposed for multi-microprocessor systems for various applications in high energy physics.[9,10] These architectures all have the common feature that they contain a number of computers connected by some form of bus to a common host node which is responsible for code development and some of the I/O handling. Each computer can talk to its own local memory and possibly to local peripherals without tying up the multiprocessor bus, and in some cases can talk to other peripherals for I/O purposes via the bus but without the aid of the host. This architecture is also that of the CM*,[11] a multiprocesor built from 50 LSI-11's at Carnegie-Mellon University to study such configurations and their operating systems. The unique feature which HEP adds to the CM* is the knowledge that either an application is consigned for all time to a particular processor because of a need to access peripherals that are only connected to that processor (e.g. distributed controls systems), or else that the computing load is naturally divided into "events" which can be distributed among the processors with a time scale which is known to the application programmer, and thus the distribution of work is never handled by the multiprocessor operating system.

Having listed the similarities among such systems, note that such systems differ in whether the slave processors execute the same instruction set as the host and in whether a particular application must be forced into a particular slave or is free to be located in any slave or set of slaves. The software developments necessary to support two particular systems will now be described.

The first system is one in which the slave processor is of a different type than the host, and the application is constrained to live in a particular slave. This is the classic case of microprocessor support using development tools on a remote system. The system in question is the controls upgrade for the SLC linac,[8] which uses Sytek system 40 as the multiprocessor bus. However, an essentially identical specification has been promulgated for a BNL-LBL-SLAC collaboration[12] which is attempting to introduce Fastbus as the multiprocesor bus to existing PEP experiments, starting with the Mark II. The software specification is as follows: (1) a FORTRAN cross-compiler implementing FORTRAN 77 extended to be essentially compatible with VAX FORTRAN will be provided; (2) a suitable cross-linker and downline loader will be provided; (3) the run-time support system at the slave node will support FORTRAN FORMAT statements (i.e. the programmer need know nothing about the internal machine representation of floating point numbers), timer services (What time is it? Execute a specified routine at a specified time. Execute a specified routing a specified time from now), and connect to interrupt services (execute a specified routine whenever a specified interrupt driven event occurs); (4) an interactive symbolic cross-debugger will be provided. The last item, which is definitely the most important, will allow a programmer sitting at a terminal on the host machine to place breakpoints in any program executing in any slave processor, and to investigate variables in the slave program by their symbolic names. It is estimated that the development of such a software support system will cost between $250K and $500K.

The second system is one in which the slave processor is assumed to have an instruction set identical to that of the host, and in which the entire set of slave processors is used to boost the processing capacity of the host system. An application can be moved from the host system into the slaves, requesting, at the time of the move, the use of any number of identical slaves available in the slave pool maintained by the host system. The use of an identical instruction set allows the debugging of any new application by developing code with the aid of the host symbolic debugger, followed by transfer to the slave processors with no code changes. A cross-debugger is not provided. The software specification allows any process running on the host machine to perform the following actions: (1) allocate a cluster of slave processors; (2) specify a program to run in any of its slave clusters; (3) get the status of a cluster; (4) wait for the status of a cluster to change (e.g. wait for slave program execution to terminate); (5) connect any logical unit (e.g. FORTRAN device 6) of any cluster of slaves to any file or device on the host, to any logical unit of the host process which owns the cluster, or to any logical unit of another cluster owned by the same host process. This system is being built at BNL to allow a VAX host system to access slave LSI VAX processors through a Fastbus multiprocessor link. A slight modification of this system may be used in the Fermilab Colliding Detector Facility to allow mangement of LSI VAX processors via Fastbus for use as a programmable trigger filter. The estimated software development cost is of order $250K.

This seems to lead us to another pair of conclusions. The first is that given a sufficient initial software effort, it seems possible to develop support tools which can manage microprocessors connected to a host system to perform essentially arbitrary tasks, as long as there is sufficient capacity on the multiprocessor bus and in the host system. In particular, it is not unreasonable to expect that after initial development costs of $100K for hardware and $250K for software, a $1M investment in microprocessors can add of order 100 MIPS of manageable processing capacity (e.g. 10 CDC 7600 equivalents) to any current online VAX system in an experimental pit. In an age of $50M experiments, this is not a large price to pay for such an outstanding amount of computing. The second conclusion concerns development of software support tools such as those mentioned in the first example above. In general, the $250K minimum software cost necessary to support a new microprocessor far outweighs the hardware development effort (estimated at $100K above), and thus again it pays even more to restrain your engineer from his desire to play with the latest chip. In the extreme, the software development cost and software production cost so far exceed the hardware cost that the method of choosing hardware is to find that hardware for which the software development costs will be minimized. Typically, a reasonable additional constraint is that the software support system be designed so as to be immediately compatible with any new hardware releases the manufacturer has in mind for the next 5-10 years.

Of course, these observations are not really new. In fact, in perusing the proceedings of the 1979 Data Acquisition Conference I noted several papers in which the management of arrays or networks of cheap computers was presented as the outstanding problem of HEP computing for the next decade.[13, 14] One speaker also indicated that all of his comments had already been made at a conference ten years earlier. The only new development is that a few attempts to build complete integrated hardware/software systems are finally in progress. However, I personally find it exceedingly distressing that, given the exceedingly large development costs for a complete system including the necessary support tools, so many distinct and noncommunicating microprocessor development projects exist in the various laboratories and universities. In my view, the high energy physics community can afford (especially in the light of the current budget problems) to support one 16 bit microprocessor system, and (at the appropriate later date) one 32 bit microprocessor system. Given the extreme difference in costs between an engineer's perception that development requires a hundred dollars or so for the cpu and a few hundred dollars at most for memory, and the total project costs including all overheads and software support of many hundreds of thousands of dollars, anyone who believes that they should start yet another microprocessor development project should be firmly directed towards employment in the private sector.

HEP Computing

Table VI presents a list of some of the computing tasks associated with high energy physics experiments. Of particular importance is the column which indicates whether a task is proportional to the number of physicists on the experiment or the volume of data taken (event size multiplied by trigger rate), or to neither of these. In the last case, the magnitude of the problem is typically still proportional to the overall scale of the experiment. A summary of microprocessor applicability to the tasks follows.

Table VI. Experimental HEP Computing Tasks

Task	Scale	Comp. Rqmt. for Hadron Collider
Control	Neither	10^4-10^5 inst./sec./application
Zero Supp.[4, 5]	Data	10^5 channels/event 10^3 events/s
Calibration[4]	Data	10^4 channels/event 10^3 events/s
Trigger[2, 15, 16]	Data	10^4 analog channels/event 10^5e/s
Event Filter	Data	10^5 inst./event 10^3 events/sec.
Online Mon.	Data	10^7 inst./event 10^0 events/sec.
Production[3]	Data	10^7 inst./event 10^8 events/year
Simulation	Data	Identical to Production
Code Dev.	People	10^5instructions/second/physicist
Physics	People	10^5instructions/second/physicist

Controls

Control of high voltage, gas systems, cryogenics, etc. was one of the first areas of experimental HEP in which single chip microprocessors were applied. These systems are typically quite limited in computational power requirements, but require special applications coding for each new usage. This is clearly an area in which the availability of a complete development package for a slave of identity distinct from that of the host, complete with interactive symbolic cross-debugging aids, would be most helpful.

Zero Suppression

Intelligent digitizers have now been around for several years, with the SLAC BADC[4] as the logical culmination of attempts by commercial manufacturers to make "smart" ADC units. Recent developments such as the FNAL TDS/RABBIT[5] system have concentrated on improving conversion speed, channel density, and dynamic range, as well as adding redundant paths for fault tolerance in applications with limited hardware accessibility. Given the need to custom design a digitizer controller, as well as frequent constraints on overall speed to minimize deadtime, this has been an area where microcoded hardware rather than general purpose microprocesors have traditionally been applied.

Calibration and Transformation

This function has frequently (e.g. BADC) been, but is not necessarily contrained to be, combined with the zero suppression function. The application typically requires a very small algorithm, and would seem to be a natural for bit-slice implementations. In general, it seems that if the functionality can be added to an existing zero suppressing digitizer, it should be, but if the task requires development of an additional microcoded calibration processor, one should do a careful analysis to see if the use of an existing packaged general purpose microprocessor coded in assembly language would be more cost effective.

Event Filtering

In this category, I include processing which makes a trigger cut based on consideration of an entire event's data buffer, rather than the restricted subset used by most trigger processors (which may even make their decisions before digitization of the majority of the data has commenced). To my knowledge, this type of processing has yet to be attempted in any large, high data rate experiment, probably because the processing power requirements are so immense, although the FNAL Colliding Detector Facility[17] is showing a strong interest in including such an option. The basic requirement here is the need to execute a large algorithm which is probably coded in FORTRAN so that personnel on shift can both understand the trigger and rapidly modify it for changing running conditions and physics needs. This would seem to require an array of 32 bit general purpose microprocessors, although a bit-slice solution along the lines of the 168/E (i.e. equipped with FORTRAN programming tools) is also a possibility.

Online Monitoring

This again requires the use of a FORTRAN coded system handling large algorithms for a sampling analysis of complete events. Recent modular software techniques which divide this analysis into a number of cooperating independent tasks, some being parts of a "standard" analysis, and some being interactive based searches for special characteristics of a restricted class of events, rather than the traditional single large "background" analysis, might slightly favor a uniform architecture cpu booster implemented in 32 bit general purpose microprocessors, which have extensive easily programmed I/O capabilities, over a microcoded emulator approach, but the latter alternative has the virtues both of being a proven performer and of having a better cost effectiveness of the final hardware.

Offline Data Reduction and Simulation

The offline production problem for the SLAC LASS experiment was the original motivation for the 168/E development project. Again, the requirements are FORTRAN and large program memories, and can be met both by 32 bit general purpose LSI microprocessors or by translated microcode bit-slice systems. Careful analysis of costs is necessary to determine whether or not the better cost efectiveness of the microcoded system is outweighed by the relative ease of programming and upward compatibility with new faster hardware releases of the general purpose microprocessor systems.

A few moments' reflection will hopefully convince you that the vast majority of the tasks which are proportional to the volume of data taken can be tackled with microprocessor based solutions, leaving only the highly interactive problems of code generation and physics results preparation for the types of processors currently in use in the field. Given that the volume of data taken in HEP experiments has been growing considerably faster than the number of physicists, microprocessors can make meeting our future computing needs considerably less painful.

Actually, bit-slice microprocessors have already found considerable use in the areas of zero suppression and data calibration, as well as in some aspects of trigger processing and data reduction. The next major advance will hopefully be the harnessing of current and future generations of complete one chip or few chip processors to the tasks of high level trigger processing, online analysis, and offline data reduction and Monte Carlo generation.

References

1. M.V. Wilkes, Manchester University Computer Inaugural Conference, July, 1951, 16 (1953); M.V. Wilkes, J.B. Stringer, Proc. Cambridge Phil. Soc., pt. 2, 49, 30 (1953).
2. T.F. Droege et al., IEEE Trans. Nucl. Sci., NS-25, 698 (1978).
3. Paul F. Kunz et al., IEEE Trans. Nucl. Sci., NS-27, 582 (1980).
4. M. Breidenbach et al., IEEE Trans. Nucl. Sci., NS-25 706 (1978).
5. T. Droege, paper presented at this conference; T.F. Droege et al., Fermilab Redundant Analog Bus Based Information Transfer system document PIN 52.
6. D.J. Nelson et al., IEEE Trans. Nucl. Sci., NS-28, 336 (1981).
7. M. Larwill et al., IEEE Trans. Nucl. Sci., NS-28, 385 (1981).
8. M. Breidenbach, private communication.
9. R. Manner, B. de Luigi, IEEE Trans. Nucl. Sci., NS-28, 390 (1981).
10. L.O. Hertzberger, Amsterdam Report NIKHEF-H/81-3 (1981).
11. R.J. Swan et al., AFIPS Conf. Proc. 46, 637 (1977); ibid, AFIPS Conf. Proc. 46, 645 (1977).
12. E. Siskind, S. Loken, M. Breidenbach.
13. Marvin Johnson, IEEE Trans. Nucl. Sci., NS-26, 4433 (1979).
14. John L. Brown, IEEE Trans. Nucl. Sci., NS-26, 4438 (1979).
15. C. Halatsis et al., CERN Report DD/79/7 (1979).
16. T. Lingjaerde, CERN Report DD/75/17.
17. A.E. Brenner et al., Fermilab Colliding Detector Facility note CDF-108; ibid, paper presented at the 1981 IEEE Nuclear Science Symposium.