A Minicomputer Microprogrammable, Arithmetic Processor

T.H. Kehl and Kenneth Burkhardt[1]
Departments of Computer Science and Physiology/Biophysics
University of Washington
Seattle, Washington 98195

Except for a few notable examples, all computers have been designed as "adder-central" architectures. "Adder-central," as used here, refers to an organization which places the Arithmetic Logic Unit (ALU) at that junction of the system through which all data must flow -- thus creating a bottleneck. In the early days, when adders were expensive, cost considerations precluded more than one ALU. Nowadays powerful ALU's are available at very low cost and a designer, even of minicomputers, can consider placing more than one ALU in a system.

Historically, in the early '60's, Seymour Cray and James Thornton(1) designed many data paths in the CDC 6600. Some were in separate functional units operating in parallel and others were in the peripheral processing units. The latter mainly handled I/O and executive functions.

These multiple data paths provided a great deal of parallelism and, as is now appreciated, improved system performance. Later the CDC-Star(2), Illiac-4(3) and TI ASC(4) all exploited multiple processors (ALU's) and gave performances unmatched by other organizations.

Each of these systems is in the "super-computer" class. Even though hardware check-out may take years, the end results can be achieved no other way. Such is not the case with minicomputers. Minicomputers have a rapid design turnover and systems have impact for only a brief period of time. Also, minicomputer systems are mass-produced and cannot be specialized for a specific kind of service.

Even though one would wish to capitalize on the mass production capabilities of computer manufacturing companies, inevitably situations arise in which specialized systems are required. At the same time advances in modern science create a constant pressure for greater arithmetic processing power. In particular, dedicated minicomputers would benefit from increased arithmetic power. For example, in the biomedical research with which we are closely associated, it is becoming quite common to use digital filters and FFT's simply to locate or screen the data. For a variety of reasons including experimental feedback, measurement accuracy, cost, etc., the best solution has been to use minicomputers connected on-line to the experiment. Although the pipeline and array processor techniques of the super computer class enhance processing speed, both are, in our opinion, too rigid for the class of problems we encounter. Rather, several criteria must be simultaneously met:

1. High arithmetic processing speed
2. High algorithmic speed, i.e., FFT's, matrix arithmetic, etc.
3. Algorithmic alterability -- microprogrammable
4. Low cost
5. Conventional, consistent design as part of a computer system
6. Maintenance ease

This paper describes a microprogrammable arithmetic processor constructed to simultaneously meet these somewhat incompatable objectives. We call the device FPLM (Floating Point Logic Machine). It is a Logic Machine (5,6,7,8) and, as such, consists of a microprogrammable control processor, one or more

[1]From a thesis submitted in partial fulfillment of the Ph.D. Supported by RR00374 from the National Institutes of Health, Division of Research Resources, Biotechnology Branch.

bidirectional buses, several functional units, and a microprogram arranged to perform a digital algorithm. Some of the FPLM's functional units are specialized for arithmetic processing but others are of a general nature. Because of the flexibility of microprogram couple with general, but simple, functional units we have been able to perform not only the usual scalar and vector/array operations, but in addition, FFT's and other "special" transformations.

FPLM Organisation

A block diagram of the FPLM's overall organization is given in Fig. 1. The control processor is strictly vertically microprogrammed. The control processor's control store consists of 8-bit words, 256 words/page with 16 pages maximum. Any combination of RAM's or ROM's may be used as long as access time is sufficiently short to sustain the 100ns (single word) and 200ns (double word) microinstruction times. Anything less than 80ns will do; we use Fairchild 93410 with about 40ns access. A μ-program counter (12 bits) controls the control store addressing, unlike a horizontally organized engine in which a field of the microinstruction contains the address of the next micro.

The FPLM has two bidirectional data buses: 1. A 16-bit bus handling the I/O and housekeeping and, 2. a 32-bit bus handling the number-manipulating functional units. The two buses are connected via a 16-to-32-bit packing register. These buses are completely separate; to the extent, that both can be active simultaneously. Separate buses also facilitate eventual use on a 32-bit computer.

To facilitate parallel operation two registers -- the instruction register and the extended program counter, store the command and its location in memory.

Although not shown in the diagram, there is a path from the host minicomputer to the control store. Using this path microprograms can be transmitted from the host computer to the FPLM. Ten signals, controlled by the host, perform the following functions:

Signal	Action
SATENLM	Set Attention Flag
CLRFLG	Clear All Error Flags
ECNT	Count FPLM μPC
EXTLD	Load FPLM μPC
EXTRST	Reset FPLM
EXTRUN	Set FPLM to Run Mode
EXTWRT	Write FPLM Control Store
SFORK	Set Fork Operation Flipflop
SJOIN	Set Join Operation Flipflop
WRTBT	Write Branch Table

To send a microprogram from the host to the FPLM the following procedure would be used:

1. Halt FPLM (micro RSTRUN)
2. Initialize PC to first address of micro program (EXTLD)
3. Write micro into that location (EXTWRT)
4. Increment μPC (ECNT)
5. If not done go to Step 3, else
6. Alter branch table (WRTBT)
7. Set μPC to starting address and
8. Start FPLM (EXTRUN).

This procedure can occur at any time, and indeed, may occur several times during the execution of a single host program. That is to say, the FPLM is dynamically microprogrammable.

Standard ROM and RAM Macroinstructions

The microinstructions allow quite independant microprogram control of the fraction, exponent, scratch pad, and ALU of the FPLM. Since some arithmetic operations are common to any computer, these have been stored in a ROM portion of control store. The operations, with timing (not including memory access) are (in μsec.):

					16-bit	32-bit
FADD	3.0	SQRT	7.0	ADD	.40	.40
FSUB	3.0	LN	10.0	SUB	.40	.40
FMY	3.5	EXP	12.0	MPY	1.4	2.8
FDIV	4.0			DIV	1.6	3.2

Most of the algorithms are quite standard; multiply shifts over strings of ones and zeros, divide is S-R-T and square root uses the "long hand" approach (9). LN and EXP use an algorithm introduced by Chen (10) which uses only adds, shifts, bit counting and a ROM for implementation. Also included in the standard set of scalar operations are FPMod, FPCompare, FIX, FLOAT, GETARG, and PUTARG.

In addition, vector operations are included: VADD, VSUB, VMPY, and VDIV. Overhead for the vector operation requires 500 ns. of houskeeping (mostly array addressing) for each element pair. Thus two vectors, A and B, each of 100 elements, added together to produce C would require: 100(3.0 +.5) + 600(1.0) = 950 μsec with a 970 ns memory. A PDP 11/45, in comparison, would require 2239 μsec with a 970 ns memory. A CDC 6400 would perform the same operation in 1090 μsec -- of course, on 60-bit words rather than 32-bit words.

Several less frequently used microroutines have also been microprogrammed and these execute from a RAM portion of control store. For example, min-max searches, series summation, cross products are resident in a library and appear to the user as any other mathematic subroutine. Although it is probably unrealistic to expect a user to do his own microprogramming, he could, if he chose, write special purpose "functional evaluators" and add these to the library.

With the addition of a quarter-wave sine table -- in ROM -- an FFT macroinstruction was implemented. The following table compares the FPLM to the identical algorithm executed on the PDP 11/20:

	PDP 11/20	FPLM
Butterfly (innerloop)	247.85	16.5μs
1024 spectral lines	1087	84.48ms
correcting 1024 array for scale factor	21.1	3.59ms
sort 1024 spectral lines to correct order (real and imaginary)	205.27	5.52ms

Thus the FPLM is more than ten-fold faster than this common computer.

Macroinstruction Format and Processing

As software has become the dominant cost factor in any computer system, special attention was given to making the macroinstruction format as simple and versatile as possible. We particularly wished to avoid the macro format of a particular computer as this would tend to limit the FPLM to that computer. Pre-fix Polish Notation meets these goals. Polish pre-fix also is the form most compilers produce at some stage of compilation.

The operation A + B → C would require 4 sequential words of memory:

```
"FADD"
 Address of A
 Address of B
 Address of C
```

A conventional 16-bit minicomputer requires at least 3 words of memory to perform this operation; of course, the addressing space of such a 3 word section of code is less than 65K. FPLM requires an additional word but has a full 65K addressing space. Vector macros use an additional word just after the instruction to indicate the vector length.

A host computer requests FPLM service by raising a flag "ATTN AUXILIARY." The FPLM is attached to the host on a DMA (Direct Memory Access Channel) and the host is halted by a request for DMA service. Additionally, the FPLM reads the host's program counter to determine the location of the instruction activating the FPLM. As long as the FPLM requests

DMA service the host computer will suspend all operations and FPLM has complete control of memory. In this manner system control is transferred from host to FPLM.

Depending on the microroutine the FPLM may either capture the host's memory or steal memory cycles. In either case the FPLM must alter the host computer's program counter to point to the next legitimate host instruction. A constant, located in the microroutine, is added to the previously acquired program counter value and the result is stored back into the host's PC.

Discussion

We view the FPLM as a segmentation of computer arithmetic outside the CPU. In general, one would expect several ALU's, each tailored to a class of tasks, to outperform a single, all-purpose adder-central ALU. Now that integrated circuits are inexpensive, even minicomputers can be constructed with multiple data processing facilities.

Beyond this rather self-evident statement is the question of how to optimally tailor the ALU. Several conflicting considerations must be kept in mind:

1. A centralised instruction stream should coordinate all of the system's tasks.
2. This instruction stream should be tampered with minimally because to do so may cause havoc with the system software.
3. Ideally one would like to add arithmetic hardware to a system with little or no disruption of service, although,
4. usually very loosely coupled hardware tends to pay heavily in speed loss.
5. An arithmetic processor should be able to evolve to include new, even highly specialized, arithmetic functions, but
6. programmable hardware tends to be slow and expensive
7. The arithmetic should be fast but
8. inexpensive. And, most important of all in our view,
9. the processor device should be part of a larger computer design strategy.

Most of these requirements have been met in the FPLM. A host's instruction stream is minimally altered with the Polish pre-fix form used. As the FPLM is a Logic Machine, and hence composed of a system of functional units, alterations and/or replacement of the functional units with improved versions can occur without disrupting service. Since the microprogram control for the FPLM is external to the host, in the worst case a whole new FPLM can be attached with only minutes' disruption. To a considerable extent new arithmetic functions can be microprogrammed and added, dynamically if the user wishes, to the FPLM repertory. Although the FPLM is only modestly fast it is quite inexpensive as compared with other special purpose arithmetic processors. As parts of a larger design strategy, several other Logic Machines have been built, including a graphics display terminal (7), an informational retrieval system (11) and a mini-computer (12).

REFERENCES

(1) J.E. Thornton, Design of a Computer: The Control Data 6600, Scott, Foresman and Co., 1970.

(2) R.G.Hine and D.P. Tate, "Control Data STAR-100 Processor Design," Proceedings of the IEEE Computer Society Conference (September, 1972), pp. 1-3.

(3) G. Barnes, R. Brown, M. Kato, D. Kuck, D. Slotnick, and R. Stokes, "The ILLIAC III Computer," IEEE Transactions on Computers, Vol. 17, No. 8 (August, 1968) pp. 746-757.

(4) W.J. Watson, "The TI ASC -- A Highly Modular and Flexible Super Computer Architecture," Proceedings FJCC (1972) pp. 221-228.

(5) T.H. Kehl, "The Logic Machine: A Modular Design System for Digital Hardware Experimentation," COMPCON '75 (February 1975) p. 305.

(6) K.J. Burkhardt and T.H. Kehl, "A Logic Machine Auxiliary Processor," COMPCON '75 (February '75), p. 309.

(7) J.Q. Torode and T.H. Kehl, "A Graphics Display Terminal Logic Machine," COMPCON '75 (February, 1975), p. 313.

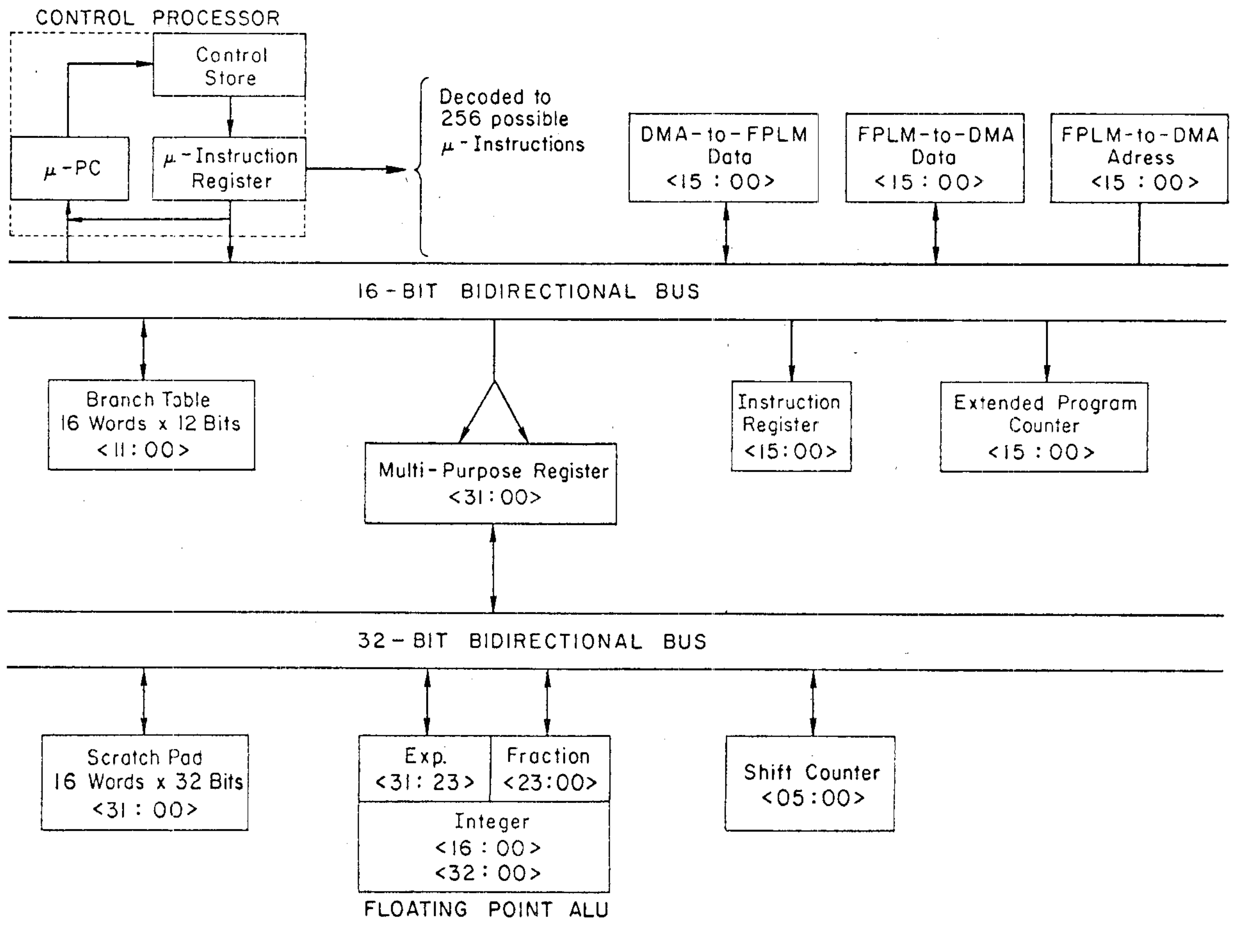
(8) J.Q. Torode and T.H. Kehl, "The Logic Machine: A Modular Computer Design System," IEEE Transactions on Computers, Vol. C-23, No. 11 (November, 1974).

(9) D. Cowgill, "Logic Equations for a Built-In Square Root Method," IEEE Transactions on Electronic Computers, Vol. 13, No. 2 (April, 1964), pp. 156-157.

(10) T.C. Chen, "Automatic Computation of Exponentials, Logarithms, Ratios and Square Roots," IBM Journal of Research and Development, Vol. 16, No. 4 (July, 1972), pp. 380-388.

(11) L.G. Berdahl, "A High Level Language Machine Design," Master's Thesis, University of Washington (July, 1974).

(12) T.H. Kehl, C. Moss, and L. Dunkel, "LM^2 -- A Logic Machine Minicomputer," IEEE Computer, in Press.


CONTROL PROCESSOR
Control Store
μ-PC
μ-Instruction Register
Decoded to 256 possible μ-Instructions
DMA-to-FPLM Data <15 : 00>
FPLM-to-DMA Data <15 : 00>
FPLM-to-DMA Adress <15 : 00>
16-BIT BIDIRECTIONAL BUS
Branch Table 16 Words x 12 Bits <11 : 00>
Multi-Purpose Register <31 : 00>
Instruction Register <15:00>
Extended Program Counter <15 : 00>
32-BIT BIDIRECTIONAL BUS
Scratch Pad 16 Words x 32 Bits <31 : 00>
Exp. <31 : 23>
Fraction <23:00>
Integer <16 : 00> <32 : 00>
FLOATING POINT ALU
Shift Counter <05:00>