SLAC-PUB--3069

DE83 012375

SLAC-PUB-3069
CERN/DD/83/3
MARCH 1983
(E/I)

THE 3081/E PROCESSOR

PAUL F. KUNZ, MIKE GRAVINA, GERARD OXOBY, AND QUANG TRANG*
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94305

and

ADOLFO FUCCI, DAVID JACOBS, BRIAN MARTIN, AND KENNETH STORR
CERN
1211 Geneva 23, Switzerland

Invited paper presented at the Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padova, Italy, March 23-25, 1983.

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

1. INTRODUCTION

Since the introduction of the 168/E ,[1-2] emulating processors have been successful over an amazingly wide range of applications.[3] For example, the 168/E has been used for off-line data processing at SLAC,[4-5] CERN,[6-8] and DESY[9] where thousands of lines of FORTRAN are involved and the processing takes many seconds per event. The same processor has been used at SLAC as a trigger processor[10-11] involving only a few hundred lines of assembly code and taking only 100 μsecs, and at CERN as a trigger processor involving hundreds of lines of FORTRAN and taking tens of milliseconds.[12] The processor has even been used for Monte Carlo lattice calculations[13] involving a few hundred lines of FORTRAN and yet taking an hour of processing time. Still more applications are planned at Saclay,[14] University of Siegen,[15] University of Toronto,[16] I.N.S.-Tokyo,[17] and at Cornell.[18]

The 168/E has its shortcomings, however, which have limited its use. This paper will describe a second generation processor, the 3081/E. This new processor, which is being developed as a collaboration between SLAC and CERN, goes beyond just fixing the obvious faults of the 168/E. Not only will the 3081/E have much more memory space, incorporate many more IBM instructions, and have full double precision floating point arithmetic, but it will also have faster execution times and be much simpler to build, debug, and maintain. The simple interface and reasonable cost of the 168/E will be maintained for the 3081/E.

The name of this processor needs a little explanation. IBM has recently come out with a new series of high performance mainframes which are called the 308x series. To the end-user, these machines have the same instruction set as the 360/370 series of machines. Our new emulating processor takes its name from the first mainframe in this series: the 3081.

2. ARCHITECTURE

The architecture of the 3081/E is shown in Figure 1. There are four execution units interfaced to two 64 bit wide busses, called the ABUS and the BBUS. There

is one for integer operations, one for floating point addition and subtraction, one for floating point multiplication, and one for division. An arithmetic operation is started by a microinstruction that transfers two operands simultaneously on the ABUS and BBUS busses to the input registers of an execution unit. The execution unit then operates on the operands internally. After enough processor cycles have elapsed for completion of the operation, the results are presented on the BBUS when a microinstruction calls for them.

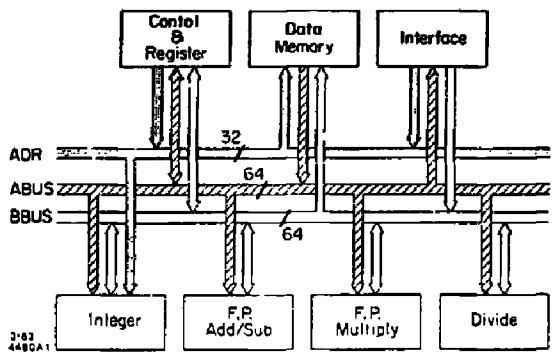


Figure 1. Block diagram of $3081/E$.

Also interfaced to these busses are the control and register unit, data memory, and the interface. The control and register unit serves three functions: it contains the microprogram address counter and conditional branching logic, the data memory address logic, and the register files.

Most IBM arithmetic instructions are of the form:

$$B \; Op \; A \rightarrow B$$

where 'B' is called the first operand and is usually a register, 'A' is called the second operand and may be either a register or memory, and 'Op' is some arithmetic operation. About 75% of the instructions encounted in execution are of the form where the second operand is memory. For this reason the data memory is interfaced to the execution units via the ABUS. If both operands come from

registers, then the control and register board supplies the second operand on the ABUS. In stores to memory, it is memory that behaves like the first operand, therefore, stores to memory are done via the BBUS. This structure allows stores to memory to be done directly from the output of an execution unit.

The design philosophy of the 3081/E processor is simplicity of design and efficiency for important instructions. Of the two, the simplicity of design can not be over-emphasized. Members of the 3081/E collaboration, and many others, have built and debugged a processor with the complexity of the 168/E. But in the environment of a High Energy Physics laboratory, we feel it is undesirable to introduce a processor of more complex design. We have noted that production of one prototype processor is only a small part of the overall effort and it is the rapid production of many processors that makes a real contribution to our respective laboratories.

An important goal of the 3081/E processor project, perhaps the most important goal, is to produce a processor that is simple, reliable, and easy to debug and maintain. To meet this goal, the design philosophy of the 3081/E is based on the following rules and guidelines:

- Separation of function to individual execution units in order to reduce the control logic.
- Use of standard TTL circuits that have 'second sources' to ensure supply of components in the future.
- Use of published maximum propagation time of every circuit in calculation of cycle time.
- Use of additional circuits, if necessary, rather than using a 'clever trick,' in order to make the design as straightforward as possible.

The choice of the architecture helps tremendously to reach these goals. It also has many additional benefits. The advantages are:

- The control logic for each execution unit is much simpler than it would be if, for example, than the control logic if all the operations were done on one board.

- With the reduction in control logic, it is much easier to analyze the circuit for its longest propagation delay path. It is therefore easier to design the processor to work in a given cycle time and to be sure that it will.

- Each execution unit can have enough board space to allow a straightforward implementation of its function, which not only simplifies the design but also allows for a circuit that optimizes the execution speed of its operation.

- With the reduction in control logic, each of the floating point execution units can have enough board space to easily allow implementation of full double precision arithmetic (REAL*8). Full double precision is not needed for the accuracy of the results, as been shown with the results of truncated double precision of the $168/E$, but it is highly desirable in order to compare results of the processor with those from an IBM compatible mainframe.

- The choice of having 64 bit wide busses allows 8 byte fetches and stores to memory in one cycle, which not only improves the double precision performance but also simplifies the control logic and data paths for transferring double precision operands to and from the execution units.

- The modular structure allows for additional execution units in the future as well as installation of improved versions of the current ones.

The disadvantage of this structure is that it requires more integrated circuits. That is, although the number of circuits in the control logic is greatly reduced, the number of circuits in the data paths is increased due to duplication of some functions. However, it is felt that circuits are not expensive compared to manpower effort and most of the manpower effort spent in debugging a processor is in areas of the control logic rather than the data paths.

3. REGISTERS

The registers must be tightly coupled to the memory addressing logic and the branching logic. For this reason all the general purpose registers are located on the control board. The physical implementation of the registers is as 16 dual-ported registers, each 64 bits wide using 16 29705 circuits, as shown in Figure 2. The 16 IBM General Purpose registers (Integer registers) are located in the first 8 locations with the least significant bit of the register address field choosing the most or least significant 32 bits of the 64 bit register. The 4 IBM floating point registers are located in the next 4 locations. Finally, 4 64 bit registers are left over for temporary storage. They can be used as some combination of 32 bit integer registers, 32 bit floating point registers, and/or 64 bit floating point registers.

Phy.Addr. Dec.	Hex	Bits 00 31	32 63
0	0	R0	R1
1	1	R2	R3
2	2	R4	R5
3	3	R6	R7
4	4	R8	R9
5	5	R10	R11
6	6	R12	R13
7	7	R14	R15
8	8	F0	
9	9	F2	
10	A	F4	
11	B	F6	
12	C	R24/F8	R25
13	D	R26/F10	R27
14	E	R28/F12	R29
15	F	R30/F14	R31

Figure 2. The 3081/E register file implementation.

There are several benefits in this implementation of the register file.

- To the processor's microcode, integer and floating registers look the same; a simplification of the control logic is achieved.
- Some integer instructions have 64 bit operands. Transfer of an even/odd register pair can be done in one cycle with this implementation since all registers can be treated as 64 bits wide. Thus, an improvement in execution speed, and a simplification of the control logic.
- The Load Multiple (LM) and Store Multiple (STM) instructions can be done 2 registers per cycle. These instructions are used for every subroutine call and can consume a lot of execution time; even more than some of the floating point instructions.
- The extra registers can be used for decoding some instructions.

4. MEMORY

Memory is one of the most important aspects of any computer or processor. For experimental high energy physics applications, the memory space of a processor must be large enough to simultaneously hold an event buffer, calibration constants, and enough working space for the event reconstruction program to operate. Modern and future detectors, especially those at colliding beam facilities, have tens of thousands of individual channels and their track reconstruction algorithms require a large amount of working space. Today, memory space is measured in units of MegaBytes, while a few short years ago only large mainframe processors had more than 1 MegaByte of real memory.

It would seem that large memory space could be most easily achieved by using the dense dynamic memory circuits that are commonly available. These circuits typically have 150 to 200 nsecs access time, 300 to 350 nsec cycle time, come in packages of 64K bits, and cost about US $1,000 per MegaByte. However, there are some problem areas in using these circuits. For example, it is not prudent to have a large memory using them without error correcting code logic. LSI circuits are now available for this logic, but the effect of implementing it is the need for

more memory chips for the error correcting code to be stored and a slow down of the memory cycle time.

Large memory space is important but the speed of the memory is equally important in High Energy Physics code. This is 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. Therefore, the overall speed of execution becomes dominated by memory access time as the execution time of arithmetic instructions tends to zero.

The memory of the 3081/E will be implemented using the less dense but faster static memory circuits. Today they have typically 55 nsec access and cycle time, come in packages of 16K bits, and cost about US $5,000 per Megabyte The 55 nsec access time of the memory circuit leads to a 120 nsec memory cycle time for the processor when one adds up the address decoding time, circuit access time, propagation time of bus buffer circuits, and minimum setup times at the destination. Compared to using the dynamic memory circuits, the use of static memory is also much simpler because there is no need for error correcting code logic or the refresh timing logic. Also a very rapid access time is achieved without resorting to a cache memory buffer as is done in many high performance computers.

The use of more expensive memory can easily be justified in many applications. For example, in a multi-processor application, if one used a processor ten times slower than the 3081/E but with memory that was 5 times less expensive, then one would need 10 of these processors to equal the throughput of the 3081/E and one would be spending twice the amount of money on memory circuits.

A 3081/E memory board will initially contain 1/4 MegaByte using 16K static memory circuits. The processor can accept a maximum of 14 memory boards or 3.5M Bytes. Today, most High Energy Physics programs, including their I/O buffers for each tape and disk file, run with less than 3.5M Byte allocation on an IBM mainframe. It is expected that 64K statics will be introduced in 1984

so by 1985 they will be reasonably priced. Their use will lower the price of the processor's memory and make it possible to have a processor with 14M Bytes.

5. MEMORY ADDRESS CALCULATION

The availablity of large memory with fast access times is only half the problem. To access it quickly one must also be able to calculate the memory address quickly.

In the 3081/*E* the problem is solved in the following way. Each micro-instruction that accesses memory has two completely independent fields. The first field controls the basic address calculation; i.e. adding the IBM 12 bit displacement field to the contents of a base register. This is denoted in the examples that follow as $D_2(B_2) \rightarrow MAR$. The second field controls the execution of an instruction. The address calculation will be done one micro-cycle ahead of the use of the memory operand. Thus, an isolated Load instruction would take two cycles as shown in the example below:

IBM	Instruction	3081/E micro-instruction	
L	3,328(13)	328(13)→MAR	
			(M)→R3

However, two Load instructions in a row would take only three cycles as shown below:

IBM	Instruction	3081/E micro-instruction	
L	3,328(13)	328(13)→MAR	
L	8,808(10)	808(10)→MAR	(M)→R3
			(M)→R8

All the IBM instructions with one operand in memory are handled in the same way. Note that this simple addressing pipelining makes the Load instruction execution effectively only one cycle of 120 nsec., which is the same amount of time that the Load instruction executes on an IBM 370/168. Stores to memory on the 3081/*E* will take the same amount of time as Loads, but on an IBM 370/168 they take twice as long because of the cache memory. The execution time of these simple instructions is important. For most programs, the execution time spent in loads and stores can exceed 30% of the total.

The implementation requires that in one cycle one has a read access to one of the General Purpose registers for address calculation while reading or writing to another register. This is done by using the same port of the register file that is used to output the contents of a register on the ABUS.

Instructions with both operands from registers require use of both ports of the 29705. However, the pipelining is maintained in the 3081/E by moving the address calculation up one cycle as is shown in the following example:

```
IBM Instruction        3081/E micro-instruction
L   3,328(13)          328(13)→MAR
LR  4,8                808(10)→MAR    (M)→R3
L   8,808(10)                         R8 →R4
                                      (M)→R8
```

There will always be available a 'slot' for the address calculation because every instruction that uses a memory operand will leave an opening for the next one.

A small fraction of the memory addressing instructions have a non zero index register, thus requiring the addition of 3 numbers to form the memory address. Rather than having the complexity of a 3 input adder and the logic to feed it with the contents of two registers, the 3081/E will take two cycles to complete the address calculation as is shown below:

```
IBM Instruction        3081/E micro-instruction
L   3, 64(9,10)         64(10)→MAR
                        MAR( 9)→MAR
                                       (M)→R3
```

Since the frequency of this type of addressing is only about 10% in typical code, the time penalty is not important. When it is heavily used in some loops the same pair of index and base registers will frequently be used more than once. If this condition occurs, the 3081/E will calculate the sum of the registers once and store the results in one of the temporary registers. Memory address calculations based on the register pair will then be done using this register, thus requiring only one cycle.

Branching breaks the addressing pipeline. The first instruction that accesses memory after a branch has been taken must take two cycles or more to complete.

However, the first memory accessing instruction after a branch instruction that was not taken may have its address calculation done in the cycle before the branch. This is because if the branch is taken, there is no harm in having loaded the memory address register with an address that will not be used, and if the branch is not taken then the memory accessing instruction can proceed.

6. FLOATING POINT

One of the important aspects of a processor for High Energy Physics is its floating point performance. However, attempts to vectorize High Energy Physics code, in order to make good use of processors with vector instructions (sometimes called array processors), have not yet proven successful. It seems that the nature of most experimental code, as it is usually written, is such that there is an equal mix of scalar add/subtracts and multiplies, with a large intermix of conditional statements. Also, most event reconstruction codes spend 30-40% of their execution time in the subroutines SIN, COS, ATAN, and SQRT alone. These subroutines use floating point heavily and even double precision arithmetic internally. Therefore, for a processor to have good performance, it should have fast execution time on individual floating point instructions.

The following sections describe each of the floating point execution units.

A. Floating Point Add/Subtract

Floating point addition and subtraction are fairly complex operations. They involve pre-normalization, addition or subtraction, and post-normalization. Since it is not possible to perform all of these operations in one processor cycle time, the add execution unit does the operation internally in two processor cycles.

Even internal to the add execution unit there is separation of function and circuits. For example, the pre- and post- normalization shifters are separate circuits, and the arithmetic units to compare the exponents for pre-normalization are separate from those to correct the exponent from post-normalization. Again, this implementation choice requires more circuits but greatly simplifies the con-

'rol logic and therefore the manpower effort.

B. Floating Point Multiply

Multiplication is a rather simple operation but takes many circuits for it to go fast. The implementation has been optimized for single precision execution time which will take two processor cycles to complete. In the first cycle, the mantissa of each operand passes through an array of 9 8×8 multiplier circuits and the partial products are stored in internal registers. In the second cycle, the partial products are summed. Post-normalization and exponent correction are accomplished during the cycle that the results are presented to the BBUS.

To implement double precision multiplication in the same way would take a considerable number of circuits, therefore, an iterative technique will be used that is reasonablely fast and does not require too many circuits to fit on one board. In the first cycle, each byte of one operand is multiplied by the least significant byte of the other in an array of 7 8×8 multiplier circuits and the partial products stored in internal registers. In the next cycle, the partial products are summed and stored in an internal accumulator register, while each byte of one operand is multiplied by the second least significant byte of the other. In the next cycle, the partial products are summed and added to the accumulator shifted by 8 bits and stored, while the next byte is in the multipliers. After 7 multiply cycles plus 1 accumulation cycle, the results can pass through the post-normalization logic and onto the BBUS.

C. Floating Point Divide

Division has traditionally been one of the slowest instructions in any processor and so it will be with the 3081/E also. It will be done iteratively, 2 bits per cycle.

7. INTEGER

The benefits of separate execution units for floating point are also extended to the integer instructions. All integer instructions will be done in the integer

execution unit. On this board there will be enough space to handle not only the 4 byte (INTEGER*4) and 2 byte (INTEGER*2) arithmetic operations, but also permit the data multiplexing required for the instructions with 1 byte operands (LOGICAL*1 and CHARACTER*n). This is especially important for implementation of the instructions required by the FORTRAN '77 compilers.

8. INSTRUCTION PIPELINING

The separation of execution units, each capable of operating on its operands internally, allows for instruction pipelining. The pipelining of memory address calculation with memory access has already been discussed, but now one is referring to the starting of a new instruction before the previous one is finished, or the overlapping of one instruction with another.

The following example is taken from actual code. The FORTRAN compiler frequently generates a sequence of instructions like LE 0,...; SE 0,...; ME 0,.... This would be translated into 3081/E microcode as shown below:

IBM Instruction	3081/E micro-instruction		
1) LE 0, 316(0,13)	1: 316(13)→MAR		
2) SE 0, 688(0,13)	2: 688(13)→MAR	(M)→F0	LE
	3:	(M)→A2 F0→A1	A_0
	4:		A_1
3) ME 0,1672(0,10)	5: 1672(10)→MAR		A_2
	6:	(M)→M2 AR→M1	M_0
	7:		M_1
	8:		M_2
	9:	MR→F0	

The Load instruction, 1), executes in 2 cycles, 1: and 2:, as has already been described in the section on memory addressing. The Subtract instruction, 2), has its memory address calculation overlapped with the actual memory access of the Load instruction in 3081/E instruction 2:. The start of the subtract occurs in 3081/E instruction 3: when the second operand is transferred from memory to the second operand input of the add/subtract execution unit (A2) and the first operand is supplied from register to the first operand input (A1). After the two cycles (A_1, A_2), 3081/E instructions 4: and 5:, the results of the subtract are ready.

The next IBM instruction, 3), uses these results and modifies them. So instead of transfering them back to floating point register 0, they are transferred from the output of the add/subtract execution unit (AR) to the first operand input register of the multiply execution unit (M1) using the BBUS. During this same cycle, 3081/E instruction 6:, the second operand for the multiply instruction is transferred from memory to the second operand input (M2) using the ABUS.

This is called instruction overlapping and it occurs very often in typical High Energy Physics code. Overlapping can occur whenever two sequential IBM instructions modify the same register. Measuring some codes show that about half of the floating point add/subtracts are followed immediately by a floating point multiply to the same register, and vice versa. Thus the design of the 3081/E 's execution units is such that their output is placed on the BBUS so that it can be used immediately as input to the next instruction.

A sequence such as the one given above is frequently followed by a similar sequence, but using a different register. Thus one would translate into 3081/E microcode as show below:

IBM Instruction	3081/E micro-instruction				
1) LE 0, 316(0,13)	1: 316(13)→MAR				
2) SE 0, 688(0,13)	2: 688(13)→MAR	(M)→F0		LE	
	3: 320(13)→MAR	(M)→A2	F0→A1	A_0	
	4: 692(13)→MAR	(M)→F2		A_1	LE
3) ME 0,1672(0,10)	5:1672(10)→MAR	(M)→A2	F2→A1	A_2	A_0
4) LE 2, 320(0,13)	6:	(M)→M2	AR→M1	M_0	A_1
5) SE 2, 692(0,13)	7:1676(10)→MAR			M_1	A_2
6) ME 2,1676(0,10)	8:	(M)→M2	AR→M1	M_2	M_0
	9:		MR→F0		M_1
	10:				M_2
7) AER 2,0	11:	F0 →A2	MR→A1		A_0
	12:				A_1
8) STE 2, 144(0,13)	13: 144(13)→MAR				A_2
	14:		AR→F2,(M)		

IBM instruction 4) does not depend on the results from instructions 1)-3). Therefore, it can be executed at 3081/E instruction 4:, which is only one microinstruction after IBM instruction 2) has started. Similarly, IBM instruction 5) can be started at 3081/E instruction 5:, since the add execution is pipelined internally.

This is called instruction pipelining. It also happens very often in High Energy Physics code. The code shown above could have been generated by a line of FORTRAN like:

$$XC = VIX*(XA - XZERO) + VIY*(YB - YZERO)$$

It is possible to do instruction pipelining with the 3081/E because the execution units operate independantly of each other. Note also that in 3081/E instruction 14:, the results of the add execution unit are stored to register and memory in the same cycle, thus effectively reducing the Store execution time to zero. Without instruction pipelining, the same sequence would have required 23 3081/E instructions, but with the pipelining it requires only 14.

When the code uses floating point heavily, the pipelining becomes extensive. This is illustrated by adding to the above sequence of instructions one that is based on floating point register 4 as is shown below:

IBM Instruction	3081/E micro-instruction				
1) LE 0, 316(0,13)	1: 316(13)→MAR				
2) SE 0, 688(0,13)	2: 688(13)→MAR	(M)→F0		LE	
	3: 320(13)→MAR	(M)→A2	F0→A1	A_0	
	4: 692(13)→MAR	(M)→F2		A_1	LE
3) ME 0,1672(0,13)	5:1672(10)→MAR	(M)→A2	F2→A1	A_2	A_0
4) LE 2, 320(0,13)	6: 404(13)→MAR	(M)→M2	AR→M1	M_0	A_1
5) SE 2, 692(0,13)	7:1676(10)→MAR	(M)→F4		M_1	A_2 LE
6) ME 2,1676(0,10)	8:	(M)→M2	AR→M1	M_2	M_0
	9: 688(13)→MAR		MR→F0		M_1
7) AER 2,0	10:	(M)→A2	F4→A1		M_2 A_0
	11:	F0 →A2	MR→A1		A_0 A_1
	12:				A_1 A_2
8) STE 2, 144(0,13)	13: 144(13)→MAR		AR→F4		A_2
9) LE 4, 404(0,13)	14:		AR→F2,(M)		
10) AE 4, 668(0,13)					

IBM instruction 10) starts at 3081/E instruction 10: and finishes with 3081/E instruction 13:. At 3081/E instruction 11:, however, is the start of IBM instruction 7) which finishes at 3081/ E instruction 14:. Thus the pipelining is so extensive that IBM instructions are being executed in a different order from the way they appear in the object code. Without instruction pipelining, this sequence would have taken 28 3081/E instructions, but with pipelining it takes only 14.

9. PERFORMANCE

To accurately predict the execution speed of the 3081/E is rather difficult, as, in common with many processors, it will depend on the 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. Two different event reconstruction codes 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 could be estimated by the assumption that pipelining occurs to such an extent that every instruction takes effectively 1 cycle. With the same samples of code, this assumption leads to execution time 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. As a check, the execution time was also calculated for a 168/E. Its time would be 149 μsecs, or 2.1 times slower than a 370/168 which is in good agreement with execution times measured on the 168/E.

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, 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

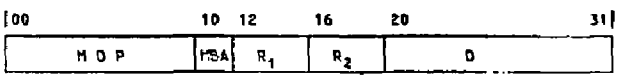
with a well known array processor. The FPS-164[19] has a theoretical maximum execution speed of 12 MFLOPS, while the 3081/E theoretical maximum is 8.3 MFLOPS. In practice, Lattice gauge programs, implemented in microcode of the array processor, achieve about 6 MFLOPS,[20] while examples of that same code, implemented in FORTRAN, would achieve 4 MFLOPS on the 3081/E.

10. THE MICROCODE AND THE TRANSLATOR

As with the 168/E, the processor's instruction set is not that of IBM's, but is its own microcode. This microcode 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. The source of the IBM object code could be the output of a compiler, or that of a linkage editor.

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 over half the total design effort of the computer. A further advantage of using the translator with the 3081/E is that instruction pipelining will be generated automatically.

The microinstruction format of the 3081/E has only two forms: register transfer instructions and conditional branching instructions. The form of the register transfer instructions is given below:



where *MOP* is a 10 bit micro operation code, R_1 and R_2 are the least significant four bits of the register addresses, *MBA* is the most significant bits of the register address, and D_2 is the displacement field for memory addressing. The *MOP* field is decoded on each board with a PROM. It controls the source for the ABUS, the source for the BBUS, the destination(s), and the length of the operands.

The form of conditional branching instructions is shown below:

where *MASK* is the IBM mask field, *tt* controls the type of branch, and the absolute branch address fills the remaining 24 bits of the instruction.

The structure of separate execution units and the pipelining of instructions at execution time has been done in large computers since the 1960's.[21--22] The difference between such computers and the 3081/*E* processor is that in a computer the pipeline has to be generated by hardware while for the 3081/*E* processor the pipeline is generated by software of the translator. Hardware generating of the pipeline can become very complex and is limited to looking ahead to a few instructions. Software generation of the pipeline is considerably easier and has no limit in looking ahead.

The 3081/*E* translator will generate the instruction pipelining and overlapping by following a simple algorithm as follows:

1. Take each IBM instruction one at a time and determine which operands are needed for execution of the instruction.
2. Starting with the previously translated instruction, scan backwards to determine where is the earliest point the execution could start. Two rules are followed to determine this point:
   (a) If a register or memory location is to be read, then find the point it was stored.
   (b) If a register or memory location is to be written, then find the point where it was last read.
3. Starting from the earliest point where the translation could take place, scan forward to the first empty microinstruction and put the microinstruction there.

This algorithm is still a one pass translation, not an optimization which would be much more difficult to program. Nevertheless, it is felt that the one pass translation yields results which are within 70-80% of maximum optimization.

11. INTERFACE

The interface to the 3081/E processor will be of the same style as the 168/E. That is, either the CPU or the interface has control of the internal busses. Thus when the processor is running, one cannot access the processor's memory from the interface. When the processor is not running, all of the processor's memory is directly addressable through the interface. From the outside, the processor will appear to be a simple slave device on a FASTBUS cable segment. The transfer rate to or from the processor could be over 64M Bytes per second if a 64 bit wide data path were used, but FASTBUS is only 32 bits wide.

There will be some improvements to make it easier to debug the processor:

- The interface will have 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 will be a stop on a Store within an address range, a stop on modification of a certain register, etc.
- The interface will be able to generate any microinstruction. This will allow the debugging of any execution unit without having the rest of the processor around.

12. CONCLUSION

The 3081/E project was formed to prepare a much improved IBM mainframe emulator for the future. Compared to the 168/E the goals for the 3081/E are:

- ***Much More Memory Space***: The advances in memory technology coming from the manufacturers now make it possible to build a 3.5M Byte processor at a cost of only US $5,000 per MegaByte while keeping the fast,

yet simple design style of the 168/E memory. Fast memory is a very important factor in a processor's speed. Large memory is needed for today's large detectors. By 1985, a 14M Byte processor should be possible at half the cost per MegaByte.

- *More IBM Instructions*: A more complete set of IBM instructions will be implemented thus allowing for use of FORTRAN '77. FORTRAN '77 is heavily used on many computers and has just recently been introduced on the IBM.
- *Full Double Precision*: REAL*8 will be handled correctly, making comparisons between output from the processor and output of an IBM computer bit for bit identical.
- *Faster Execution Times*: 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 4 times more powerful than the VAX 11/780, and 5 processors in a system would equal the performance of the IBM 3081K.
- *Less Technical Effort*: The design of the processor will be much simpler than the 168/E. The design rules will be much more conservative and will use only off-the-shelf multiple source TTL components. Every effort is being made to reduce the man-power effort to build, debug, and maintain the processor.
- *Efficient Translation to Microcode*: The translation of IBM native instructions to microcode of the processor will be maintained. It is an important element in keeping the hardware simple and fast. With the 3081/E, the translator will also automatically produce pipelined floating point operations, thus enhancing the performance for heavy floating point code.
- *Reasonable Cost and Effort*: The cost of the CPU has been considered as less of a concern than manpower effort. Nevertheless, the cost of the processor, power supply, and chassis is expected to be under US $10,000 excluding the cost of memory.

- *Simple Interfacing*: We will maintain the simple interface of the 168/E. That is to say, the processor will look like a slave on a FASTBUS cable segment.

The project is being carried out as a collaboration between SLAC and CERN DD division. At this date we have detailed block diagrams of the entire processor, simulation programs of some parts, an approximate circuit count and costs, approximate board layouts, existence proof of the translator's pipelining capabilities, and partial computer based documentation. It is planned during the calendar year 1983, that a prototype processor will be built with the work being divided equally between SLAC and CERN. Final debugging should occur at SLAC early in 1984 with processors being generally available for use by the end of 1984.

REFERENCES

1. P. F. Kunz, *The LASS hardware processor*, Nucl. Instrum. Methods *135*, 435 (1976).
2. P. F. Kunz, R. N. Fall, M. F. Gravina, H. Brafman, *The LASS Hardware Processor*, 11th Annual Microprogramming Workshop, Pacific Grove, CA, November 19-22, 1978. SIGMICRO Newsletter *9*, 25 (1978).
3. P.F. Kunz, *Use of Emulating Processors in High Energy Physics*, Proceedings of the International Conference on Experimentation at LEP, Phys. Scr. *23*, 492 (1981).
4. P. F. Kunz, R. N. Fall, M. F. Gravina, J. H. Halperin, L. J. Levinson, G. J. Oxoby, Q. H. Trang *Experience Using the 168/E Microprocessor for Off-line Data Analysis*, IEEE Trans. *NS-27*, 582 (1980).
5. L. S. Rochester, *Microprocessors in Physics Experiments at SLAC*, Topical Conference on Application of Microprocessors to High Energy Physics Experiments, Geneva, Switzerland, May 4-6, 1981. CERN 81-07 204 (1981).
6. C. Bertuzzi, D. Drijard, H. Frehse, P. Gavillet, R. Gokieli, P. G. Innocenti, R. Messerli, G. Mornacchi, A. Norton, J. P. Porte, *On-Line Use of the 168/E Emulator at the CERN ISR SFM Detector*, Topical Conference on Application of Microprocessors to High Energy Physics Experiments, Geneva, Switzerland, May 4-6, 1981. CERN 81-07 329 (1981).
7. D. R. Botterill, A. W. Edwards, *Experiences Using the 168/E Microprocessor within the European Muon Collaboration (EMC)*, Topical Conference on Application of Microprocessors to High Energy Physics Experiments, Geneva, Switzerland, May 4-6, 1981. CERN 81-07 336 (1981).
8. D. Lord, P. Kunz, D. R. Botterill, A. Edwards, A. Fucci, G. Lee, B. Martin, G. Mornacchi, P. Scharff-Hansen, M. Storr, T. Streater, *The 168/E at CERN and the MARK II: An improved processor design*, Topical Conference on Application of Microprocessors to High Energy Physics Experiments, Geneva, Switzerland, May 4-6, 1981. CERN 81-07 341 (1981).

9. T. Barklow and G. Wolf, Private Communication
10. D. Bernstein, J. T. Carroll, V. H. Mitnick, L. Paffrath, D. B. Parker, *SNOOP Module CAMAC Interface to the 168/E Microprocessor*, IEEE Trans. *NS-27*, 587 (1980).
11. J. T. Carroll, J. Brau, T. Maruyama, D. B. Parker, J. S. Chima, D. R. Price, P. Rankin, R. W. Hatley, *On-Line Experience with the 168/E*, Topical Conference on Application of Microprocessors to High Energy Physics Experiments, Geneva, Switzerland, May 4-6, 1981. CERN 81-07 501 (1981).
12. J. T. Carroll, M. DeMoulin, A. Fucci, B. Martin, A. Norton, J. P. Porte and K. M. Storr, *Data Acquisition Using the 168/E*, Paper submitted in these proceeding.
13. J. E. Hirsch, R. L. Sugar, D. J. Scalapino, R. Blankenbecler, *Monte Carlo Simulations of One-Dimensional Fermion Systems*, NSF-ITP-82-44.
14. J. Prevost, Private Communication.
15. M. Rost, *Use of the 168/E Processors as Stand Alone Computing Facilities*, Nucl. Instrum. Methods *202*, 445 (1982).
16. B. Shepard, Private Communication.
17. K. Ukai, Private Communication.
18. C. Bebek, Private Communication.
19. Floating Point Systems, Beaverton, Oregon.
20. K. Wilson, Private Communication.
21. J. E. Thornton, *Design of a Computer, The Control Data 6600*, Scott, Foresmand, and Co., Glenville, Illinois(1970).
22. R. M. Tomasula, *An Efficient Algorithm of Exploiting Multiple Arithmetic Units*, IBM Journal of Research and Development, *11*, 25 (1967).