Version 2.1/UPR-0184E

Initial Experience with the Intel *i*860 Microprocessor

L.D. Gladney, P.T. Keener, N.S. Lockyer, K.J. Ragan

David Rittenhouse Laboratory, University of Pennsylvania, Philadelphia, PA, 19104

J.G. Heinrich, K.T. McDonald

Joseph Henry Laboratories, Princeton University, Princeton, NJ, 08544

March 12, 1990

Abstract

Scalar performance of the 33Mhz Intel *i*860 microprocessor is presented. Standard High Energy Physics programs have been ported to the *i*860 environment. Benchmarks have been performed and the problems encountered are discussed. Testing of the vector features of the *i*860 has recently started.

Introduction

The Intel *i*860[1] microprocessor utilizes VLSI hardware design ($> 10^6$ transistors) with 64-bit RISC architecture to provide high computational throughput. This is the first in a family of processors being developed by Intel that is expected to achieve 200 MIPS per processor by the year 1998.

Our interest[2] in the *i*860 is enhanced by an ongoing project at Intel Scientific Computers to produce a processor-farm architecture using the *i*860 with 200 Mbyte/sec data paths between nodes.[3] Such an architecture, based on an array of high performance processors interconnected with high bandwidth data paths, is well suited to both online and offline computing needs of the next generation[4] of High Energy Physics (HEP) experiments.

[1] *i*860, 386, and 387 are trademarks of Intel Corporation.

[2] L.D. Gladney *et al.*, *Proposal to the SSC Laboratory for Research and Development of a Parallel Computing Farm* (Oct. 1989).

[3] see, for example, Federal Computer Week, Vol. 3, No. 15 (April 10, 1989).

[4] The BCD Collaboration, *An Intermediate and Low p_T Dectector for the SSC*, SSC-240.

The *i*860 includes on a single IC:

- an Integer Unit,
- a Floating Point Unit,
- a Graphics Unit,
- a Memory Management Unit,
- an 8 Kbyte Data Cache, and
- a 4 Kbyte Instruction Cache.

The integer unit provides instruction flow control as well as integer arithmetic operations. The data cache provides support for vector operations; the programmer can explicitly use the data cache as if it were a large block of vector registers. The architecture provides 32 user registers for 32-bit integer operations and a separate set of floating point registers which can be used as either 32 single-precision (32 bit) or 16 double-precision (64 bit) registers. The floating point registers are also used for graphics operations. The graphics unit provides specialized instructions that implement advanced 3D drawing algorithms more effectively than previous processors.

The *i*860 provides two types of parallelism. In the first, the processor can be made to overlap floating point operations with administrative functions (*i.e.* program control) to allow two instructions per clock cycle. The second method involves pipelining certain floating point instructions, such that the processor can execute an add and a multiply simultaneously, again allowing two instructions per clock cycle. By combining these two pipelining methods, three instructions per clock cycle can be attained for limited sets of instructions.

The performance of the *i*860 RISC microprocessor has been studied. We obtained an *i*860 simulator with full software support in March 1989, and a development system, including an *i*860 board installed in a 386 host, has been operating since October 1989.

Development System

The development system, called the Star860, consists of an Intel System 302 with 25 MHz 386 DX microprocessor with 387 DX math co-processor, 33 Mhz *i*860 CPU add-in card with four Mbytes of local memory, UNIX[5] System V/386 R3.2 operating system with TCP/IP network software, and complete *i*860 development software, including C and Fortran compilers, Fortran vectorizer, assembler, linker, simulator, debugger, and complete run-time libraries. The C and Fortran compilers were written by Green Hills Software, Inc. and the vectorizer was written by Pacific-Sierra Research Corporation.

To execute a user program on the *i*860, the code must first be downloaded to the *i*860 memory from the 386 host. The 386-resident program that downloads the code then communicates with

[5]UNIX is a registered trademark of AT&T.

a small kernel executing on the *i*860. This kernel passes all system calls (*eg.* I/O) to the 386 for execution and the user program on the *i*860 halts until the 386 system call returns.

The debugger provided for the *i*860 is similar to *adb*, a standard machine-level debugger that is provided with most UNIX machines. The debugger provides for inspection and modification of processor registers, data in core memory, and disassembled instructions. It has a simple source-file interface that will display the source line in which a fatal trap occurs and all the assembly statements for any given line of source.

Test Program

The testing of the *i*860 has two objectives. First, we want to examine the difficulties in porting typical High Energy Physics code to the *i*860. Code developed for High Energy Physics has historically been difficult to port for various reasons including lack of coordination among authors, insufficient use of 'standard' coding practices, heavy use of bit manipulations, *etc.*, and for these reasons constitutes a severe test for a new compiler. Porting such code to the *i*860 can thus aid in identifying and eliminating software and hardware bugs in the compilers and the microprocessor itself, and will help us to gain a better understanding of the features of the *i*860, thereby easing the porting of code in the future.

The second goal is to obtain benchmarks that compare the performance of the *i*860 to other processors running the same physics code. Although we are concentrating on code actually used in physics event simulation and data analysis, we have also used standard benchmarking programs for comparison to other processors.

Code Porting and Problems

The standard benchmarking programs used were Dhrystone, versions 1.1 and 2.1, and Whetstone, both single- and double-precision versions. The Dhrystone benchmark[6] is a synthetic program using primarily integer computation; the Whetstone[7] is a synthetic mix of integer and floating point calculations using function calls, array indexing and conditional jumps. Because of this mix, it is expected that the Whetstone more accurately predicts the performance of a microprocessor for real HEP code.

The High Energy Physics programs and libraries that we have been working with consist of three Fermilab benchmarks (`RECON1, RECON2,` and `QCD`) and the physics simulation packages `ISAJET` and `GEANT`, which in turn use the packages `HBOOK` and `ZEBRA` and the libraries `KERNLIB` and `GENLIB`. The `GEANT` package has been used from several different driver programs.

`RECON1` is a simulated track reconstruction program, whereas `RECON2` is a real track reconstruction program. Both programs were modified at Fermilab to read from pre-initialized data in common blocks rather than from disk files. This was done to avoid the overhead of disk I/O

[6] R.P. Weicker, *Comm. ACM* **27**, *1013(1984).*
[7] H.J. Curnow and B.A. Wichman, *Computing Journal* **19**, *43(1976).*

and thus better measure processing speed. `QCD` is a lattice gauge calculation program and is very compute-intensive, spending approximately 80% of its time in three calculational subroutines.

The only problem encountered in porting this code was in trying to vectorize the Fortran. We believe that the problem is caused by a bug in the vectorizer, which will produce function calls to nonexistent vector primitives when it finds a specific instruction in a loop that it is attempting to vectorize. Although this could possibly be fixed by rewriting small sections of the code, we are anxiously awaiting a beta release of the vectorizer.

`GEANT` and its associated libraries were extracted from Patchy[8] `PAM` files, by specifying them to be compiled for the VAX/VMS[9] environment. The problems found can be classified as being one of three types: compiler bugs, machine-specific routines that needed to be written for the *i*860, and compiler and processor differences such as data types and floating-point representation.

Compiler Bugs

We used Version 1.8.5β of the Green Hills Fortran compiler. Most of the porting difficulties were caused by three distinct compiler bugs. The first is a problem that appears when a `REAL*4` function contains a `REAL*8 ENTRY` point. The result can be anything from floating-point exceptions (arising from the use of 32 bits of a 64-bit quantity) to output that will not assemble. This is similar to a well-known bug found in the MIPS Fortran compiler. The work-around used was to split the function in two, with the single-precision version calling the double-precision version and explicitly performing the conversion back to single precision.

The second bug is associated with the VMS-compatibility compiler option. The VMS compiler has an intrinsic called `ISHFT(I,J)` that will shift word `I` by `J` bits to the left or $-$`J` bits to the right. The right shift is an arithmetic shift, *i.e.* it is sign-extended. The implementation of this intrinsic in the *i*860 Fortran compiler is not compatible with the VMS compiler in that if J is a variable rather than a constant, the right shift is done as a logical shift. There were a small number of cases (approximately 20 in the `KERNLIB` library) in which this was observed to be a problem. Code was written to work around the deficiencies of the Fortran compiler to correct for this.

A third compiler bug relates to use of external function calls in statement functions. The compiler fails with an internal compiler error. This problem was communicated to Intel and acknowledged as a compiler bug. It should be fixed in the next release of the compiler.

Machine-Specific Routines

The second type of problem encountered was with machine-specific routines that had to be written or adapted for the *i*860. These were generally bit manipulation or error-handling routines; they were modified to work properly with the *i*860 compiler and hardware, or avoided when not absolutely needed for proper execution.

[8]Patchy is a CERN-written code management system.
[9]VAX and VMS are trademarks of Digital Equipment Corporation.

Compiler and Processor Differences

The third type of problem involved issues like lack of support for hexadecimal constants and `REAL*16` quantities in the compiler, and slight but significant differences in the range of floating-point types from the VMS standard. These problems were fixed by minor recoding.

Results and Discussion

We have made significant progress in our porting efforts. We have generated `ISAJET` events and the testing programs for the `ZEBRA` and the `GENLIB` library have all performed successfully. We have generated histograms with the `HBOOK` package. Most of the `KERNLIB` routines pass the testing program. We have been able to generate events with `GEANT` version 3.12, and are in the process of obtaining benchmarks.

We have run the standard suite of CPU benchmarks, the Fermilab benchmarks, and have compared the *i*860 performance running `ISAJET` to that of other processors. The results are **preliminary**, and all *i*860 results were obtained without hand optimization or vectorization. Compiler optimization was turned on for the *i*860 runs, and was set at level 2 for the DecStation 3100 results and at level 3 for the IBM and Amdahl results.

Our results for the various programs of the standard benchmark suite follow:

Program	Result
Dhrystone V1.1	72600 dhrystones/second
Dhrystone V2.1	67300 dhrystones/second
Single-precision Whetstone	26500 whetstones/second
Double-precision Whetstone	20000 whetstones/second

Table 1: Results of the standard benchmark suite for the *i*860.

Our results for the benchmark using `ISAJET` follow. These results refer to the generation of 1000 QCD $b\bar{b}$ events. For this comparison, all event `I/O` was suppressed in order to compare only computational speed.

Machine/Processor/Compiler	Time (sec)
VaxStation 3100/What processor, what compiler ?	409
DECstation 3100/MIPS R2000 (16 MHz)/F77	129
IBM/3090-200E(VF)/FortVS2 (Release 4)	27
Amdahl/5890-600/FortVS2 (Release 3)	22
*i*860 (33 MHz)/Green Hills V1.8.5	61

Table 2: Comparative results running `ISAJET` on various processors.

Results from the Fermilab benchmarks `RECON1, RECON2`, and `QCD` are summarized in tables 3, 4 and 5, respectively. The first column in the tables gives the name of the processor and the compiler

used. The second column is the execution time for the program. The third and fourth columns are the speed-up factors of the given CPU (i.e. the inverse of the execution time) normalized to the VAX 11/780 (compiler version 3) and VAX 11/780 (compiler version 4) execution times, respectively. The execution times in tables 3–5 for the VAX and Motorola microprocessors were supplied by I. Gaines of Fermilab.

CPU/Compiler	Time (sec)	VAX V3	VAX V4
VAX 11/780 /Version 3	47	1	0.70
VAX 11/780 /Version 4	33	1.42	1
Motorola 68020/Absoft	65	0.72	0.51
Motorola 68020/Green Hills	47	1.00	0.70
*i*860/Green Hills V1.8.5	3.4	13.8	9.70

Table 3: Results of running the benchmark `RECON1` on various combinations of processor and compiler.

CPU/Compiler	time (sec)	VAX V3	VAX V4
VAX 11/780/Version 3	139.5	1	0.76
VAX 11/780/Version 4	106.1	1.3	1
*i*860/Green Hills V1.8.5	14.5	9.6	7.3

Table 4: Results of running the benchmark `RECON2` on various combinations of processor and compiler.

CPU/Compiler	Time (min)	VAX V4
VAX 11/780/Version 4	11	1
Motorola 68020/Absoft	54	0.20
*i*860/Green Hills V1.8.5	2.43	4.52

Table 5: Results of running the benchmark `QCD` on various combinations of processor and compiler.

The standard Dhrystone and Whetstone benchmark results of Table 1 are compatible with the numbers published by Intel for the *i*860[10], given the differences in test systems. These numbers show that the 33 MHz *i*860 performs at approximately 25 MIPS (where we take the VAX 11/780 as a 1-MIPS machine). On a real HEP application, Table 2 shows that the *i*860 performs a factor of ≈ 2.1 times faster that the DecStation 3100. Intel's comparison of the *i*860 and the MIPS M/1000 system[11], which has the same CPU and clock rate as the 3100, implies that the speed-up factor should be closer to 2.5. We are currently trying to resolve the differences between these numbers, and to further explore the vectorization of these HEP codes on the *i*860.

[10] *i*860 *Processor Performance*, Release 1.0, March 1989, Intel Corporation.
[11] *ibid.*