VLSI Implementations of a Reduced Instruction Set Computer

Daniel T. Fitzpatrick, Manolis G.H. Katevenis, David A. Patterson, Zvi Peshkess, Robert W. Sherburne, John K. Foderaro, Howard A. Landman, James B. Peek, Carlo H. Séquin, and Korbin S. Van Dyke

University of California at Berkeley
Computer Science Division/EECS Department
Berkeley, California 94720

1. INTRODUCTION

A general trend in computers today is to increase the complexity of architectures commensurate with the increasing potential of implementation technologies. Consequences of this complexity are increased design time, more design errors, inconsistent implementations, and the delay of single chip implementation[7]. The Reduced Instruction Set Computer (RISC) Project investigates a VLSI alternative to this trend. Our initial design is called RISC I.

A judicious choice of a small set of the most often used instructions combined with an architecture tailored to efficient execution of this set can yield a machine of surprisingly high throughput. In addition, a single-chip implementation of a simpler machine makes more effective use of limited resources such as the number of transistors, area, and power consumption of present-day VLSI chips[8]. Simplicity of the instruction set leads to a small control section, a comparatively short machine cycle, and a reduced design cycle time.

Students taking part in a multi-term course sequence designed two different NMOS versions of RISC I. The "Gold" group (Fitzpatrick, Foderaro, Peek, Peshkess, and Van Dyke) designed a complete 32-bit microprocessor, currently being fabricated. The "Blue" group (Katevenis and Sherburne) started from the same basic organization but introduced a more sophisticated timing scheme in order to shorten the machine cycle and also reduce chip area. At present, only the data path of this more ambitious design has been completed. The chips were designed using only horizontal and vertical lines ("Manhattan" design), with the simple and scalable Mead-Conway design rules (fabrication: $\lambda = 2$ microns, no buried contacts).

At the onset of the design of RISC I we defined the following goals and constraints: (a) find a reasonable compromise between high performance for high-level language programs and a simple, single chip implementation; (b) make the size of all instructions equal to one word and execute all instructions in one machine cycle; (c) emphasize register oriented instructions and restrict memory access to the LOAD and STORE instructions. The resulting architecture has 31 instructions in two formats, uses 32-bit addresses, and supports 8-, 16-, and 32-bit data.

The chip area saved by simplicity of the control circuitry was devoted to a very large set of 32-bit registers. This permits the processor to allocate a new

set of registers for each procedure call and thus avoids the overhead of saving registers in memory. By overlapping these windows of registers, parameters may be passed to a procedure by simply changing a pointer.

This so called overlapped register window scheme[9] is largely responsible for the surprisingly good performance obtained in simulation of high-level language programs. Simulations of benchmark programs written in 'C' indicate that RISC I can run faster than many commercial minicomputers. Table 1 shows the size and execution time of six C programs on RISC I assuming a machine cycle of 400 nanoseconds, and 250 nsec instruction-fetches. Also in the table is the VAX 11/780, a 32-bit Schottky-TTL minicomputer with a 200 ns microcycle time; and the Z8002, a 16-bit NMOS microprocessor with a microcycle time of 250 ns. Even though the Z8002 is using only 16-bit addresses and data while RISC is using 32-bit addresses and data, RISC programs are typically 10% larger while running about four times faster. The byte-variable length of VAX instructions reduces program size by about a third; on the other hand, every C program that we have run on the RISC simulator has been faster than the on VAX.

In addition to good performance, in this paper we show that the design of RISC I also was several times faster and required only one-fifth the manpower of comparable machines. The most visible impact of the reduced instruction set is that the area dedicated to control logic has dropped from 50 % in typical commercial microprocessors to only 6 % in RISC I.

Table 1.
RISC Program Size and Execution Times
Relative to a VAX 11/780 and a Z8000

	Program Size (bytes)					*Execution Time (secs)*				
Name	RISC	VAX	VAX / RISC	Z8000	Z8000 / RISC	RISC	VAX	VAX / RISC	Z8000	Z8000 / RISC
acker	208	120	0.58	238	1.15	3.2	5.1	1.6	6.8	2.8
qsort	644	436	0.68	648	1.01	0.8	1.8	2.3	4.7	5.9
puzzle(sub)	2468	1668	0.68	1812	0.7	4.7	9.5	2.0	19.2	4.2
puzzle(ptr)	2480	1700	0.68	1656	0.7	3.2	4.0	1.3	7.5	2.3
sed	17368	14336	0.83	17500	1.01	5.1	5.7	1.1	22.2	4.4
towers	132	100	0.76	242	1.82	6.8	12.2	1.8	28.7	4.2
Average	3883	3060	0.7 ±.1	3649	1.1 ±.4	4.0	6.4	1.7 ±.4	15.2	4.0 ±1.2

2. MICRO-ARCHITECTURE

The simplicity and regularity of RISC permits most instruction executions to follow the same basic pattern: (1) read two registers, (2) perform an operation on them, and (3) store the result back into a register. Jump, call, and return instructions add a register (possibly PC) and an offset and store the result into the appropriate PC latch. The load and store instructions violate the original constraint: in order to allow enough time for access of the main memory, they add the index register and immediate offset during the first cycle, and perform the memory access during an additional cycle. In all cases, while the processor is executing the first cycle of an instruction, the next instruction is fetched from memory.

The micro-architectures of the two implementations are tailored according to the above characteristics. The CPU can be subdivided naturally into the following functional blocks: the register-file, the ALU, the shifter, a set of Program Counter (PC) registers, the Data I/O latches, the Program Status Word (PSW) register, and control (which contains the instruction register, instruction decoder, and internal clock circuits). Since two operands are required simultaneously, the register file needs at least two independent busses and a two-port cell design. For speed, the registers are read with dynamically precharged bit lines. This requires the following basic timing sequence: (1) register read, (2) arithmetic/logic/shift operations, (3) register write, and (4) bus precharge for next read. The cycle time is determined by this sequence of operations. For the price of a third bus, phase (4) can be eliminated: as the result is written back into the register file by this extra bus, the two read busses are precharged for the following read phase. This simple but robust 3-phase scheme has been adopted in the Gold processor. The basic organization with busses A,B (read only) and C (write only) is shown in Figure 1. It permits simple instruction fetch and execution.

During progress of the Gold RISC I design, it became apparent that a three-bus register cell incurred a significant area penalty. Since a large fraction of the chip area is devoted to the register file, more attention was focused on the design of a compact bit cell. The classic six-transistor static RAM cell was chosen for its compactness in the second, more ambitious Blue chip. Reading is accomplished by selectively discharging the precharged bit line busses. Contrary to commercially available static RAMs, no sense amplifiers are used, yielding a speed penalty. However, a two-port reading capability is obtained. Writing is accomplished by putting both the data and its complement onto the two busses, as in a typical static RAM. Before reading, the busses must be precharged for proper operation.

The Blue RISC I design (Figure 2) was based on this two-bus, two-port register cell. The reduced cell size allowed significant performance improvement due to the shorter RC delay in the polysilicon control lines running across the data path. The overall chip size was also considerably reduced.

Further improvements were made by overlapping the register-write with the execution of the following instruction. The result of an operation is kept in a

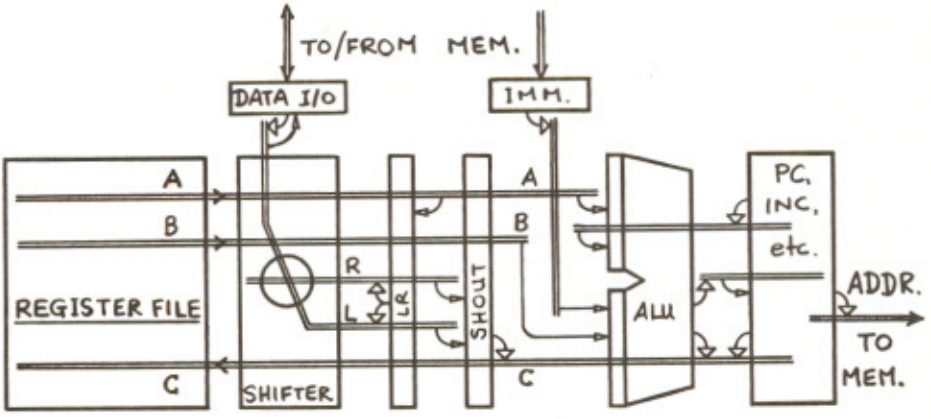

figure 1 : The Gold Data-Path.

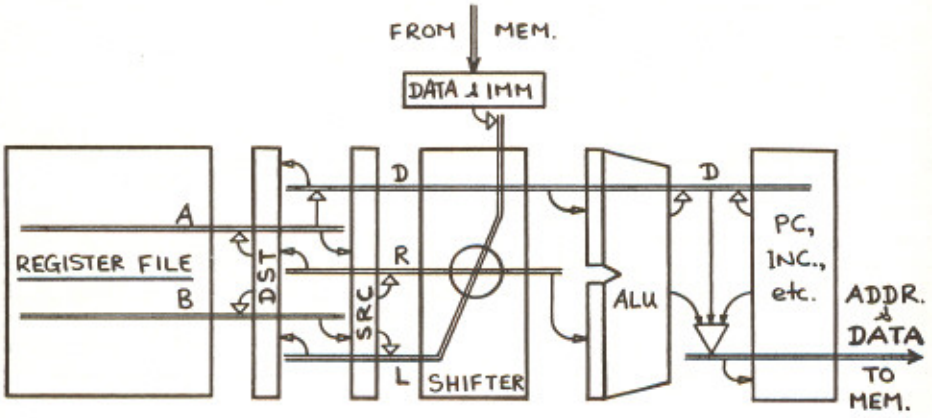

figure 2 : The Blue Data-Path

temporary latch (DST), and is only written into the register-file during the operation-phase of the next cycle. Special "internal forwarding" circuitry takes care of the instructions that use the result of their immediately previous one. In effect, each instruction now requires three cycles: (1) Instruction fetch and decode; (2) Register read, operate, and temporary latching of result; (3) Write result into register file. However, in the Blue design, these three operations are pipelined so that a new instruction begins each cycle (except load/stores).

Both designs multiplexed the address and data pins, as we could not afford to use 64 separate lines with current packaging technology. Power consumption for the Gold chip is estimated to be between 1.2 and 1.9 Watts.

3. CHIP ANALYSIS

We have analyzed the completed Gold chip and the data path of the Blue chip. Table 2 shows how the chip resources are allocated for the various functional blocks in the Blue and Gold designs (figures for Blue control and I/O are estimates). These resources are measured in million square lambda, thousands of transistors, and thousands of rectangles. We can make several observations based on this table:

(1) *"Control" is only 6 to 8 % of the total chip.* Even if there were no registers, it would still be only 12 %. Section 5 compares this important result to commercial microprocessors, and discusses its significance.

(2) *The Blue data path* is considerably more compact than the Gold data path. The register file pitch (42 λ in Blue, 77 λ in Gold) determined the height of the data path in both versions; the rest of the modules were designed accordingly. The compactness of the cross coupled static RAM cell allowed the 138 32-bit registers of the Blue design to occupy about half the area of the 78 32-bit registers of the Gold design.

(3) *SCAN IN/SCAN OUT (SISO) is less than 5 % of the chip.* SISO is a technique which improves chip testability by allowing access to each state bit in a module. The flip-flops are connected together as a long shift register, allowing serial reading and writing. The Gold chip has complete SISO on the shifter, ALU, control, and some of the PC registers. As we had spare pins we used 11 pins for SISO (4 % of chip area); we could have used fewer.

Table 2.

Area, transistors, and rectangles per Blue and Gold functional block.

functional block	AREA (M λ^2)				FUNCTION (K transistors)				COMPLEXITY (K rectangles)			
	Gold		Blue		Gold		Blue		Gold		Blue	
Registers	7.72	39 %	4.12	30 %	30.05	68 %	26.50	66 %	372.2	69 %	300.0	65 %
register decoder	1.56	8 %	0.87	6 %	3.33	7 %	3.60	9 %	30.6	6 %	40.0	9 %
Shifter	1.89	10 %	0.89	6 %	2.63	6 %	2.93	7 %	39.3	7 %	39.4	9 %
ALU	1.41	6 %	0.44	3 %	3.06	7 %	2.26	6 %	32.3	6 %	23.3	5 %
PC's	0.86	4 %	0.36	3 %	1.98	6 %	1.64	4 %	20.2	4 %	16.6	4 %
Data I/O Logic	0.23	1 %	0.10	1 %	0.67	2 %	0.37	1 %	6.5	1 %	3.5	1 %
Scan In Scan Out	0.14	1 %	0.06	0 %	0.38	1 %	0.14	0 %	1.6	0 %	1.2	0 %
total DATA PATH	13.84	69 %	6.84	50 %	42.09	95 %	37.44	94 %	502.8	93 %	424.0	92 %
PLA's	0.23	1 %			0.84	2 %			5.5	1 %		
Latches	0.42	2 %			0.83	2 %			8.5	2 %		
Routing	0.47	2 %			-	-			6	1 %		
Scan In Scan Out	0.04	0 %			0.10	0 %			0.6	0 %		
total CONTROL	1.16	6 %	~1.16	~8 %	1.77	4 %	~1.66	~4 %	20.6	4 %	~20.0	~4 %
Routing	2.05	10 %	~2.75	~20 %	-	-	-	-	3.8	1 %	~5.0	~1 %
Pads	1.13	6 %	~1.10	~8 %	0.84	2 %	~0.84	~2 %	9.9	2 %	~10.0	~2 %
Scan In Scan Out	0.73	4 %	~0.65	~5 %	0.06	0 %	~0.06	~0 %	1.2	0 %	~1.0	~0 %
total I/O	3.91	20 %	~4.50	~33 %	0.92	2 %	~0.90	~2 %	14.9	3 %	~16.0	~4 %
Unused area	1.09	6 %	~1.20	~9 %	-	-	-	-	-	-	-	-
TOTAL CPU	20.00	100 %	~13.70	100 %	44.42	100%	~40.00	100 %	538.3	100%	~460.0	100 %

(4) *The number of transistors relative to that of rectangles, in each functional block, is strongly correlated;* there exist about ten rectangles for each transistor.

(5) *Area utilization varies strongly.* The highly regular data path, consisting mainly of carefully optimized cells, contains more than 90 % of the transistors and rectangles but only half to two-thirds of the area. On the other hand, the I/O area, which is dominated by random interconnections, contains 20 to 30 % of the area but less than 4 % of the transistors or rectangles.

4. TOOLS

There is no doubt in our minds that the key to successful VLSI design lies in appropriate software. While we have plans for a sophisticated design environment[5], for the moment we have to work with a rather small subset. We used more than a dozen programs to design our chips. Among those, we felt that the following six tools were invaluable - we cannot imagine how we would have been able to get this far without them:

CAESAR	color graphics editor	John Ousterhout	U.C. Berkeley
CIFPLOT	plot of mask layers	Dan Fitzpatrick	U.C. Berkeley
MEXTRA	Manhattan circuit extractor	Dan Fitzpatrick	U.C. Berkeley
SLANG	multi-level simulator	John Foderaro	U.C. Berkeley
MOSSIM	switch level simulator	Bryant/Terman	M.I.T.
DRC	layout rules checker	Clark Baker	M.I.T.
MKPLA	PLA generator	Howard Landman	U.C. Berkeley

Other tools used to some degree and which were particularly helpful for our project include:

PRESTO	PLA minimizer	Fang/Newton	U.C. Berkeley
EQNTOTT	PLA equation translator	Robert Cmelik	U.C. Berkeley
SPICE	circuit simulator	Pederson/Newton	U.C. Berkeley
STAT	electrical rules checker	Forest Baskett	Stanford/Xerox PARC
POWEST	power estimator	Robert Cmelik	U.C. Berkeley

A key factor was the glue that held all these various tools together and produced a cohesive design environment; this function was provided by the UNIX operating system (4th Berkeley Software Distribution) [2] running on a DEC VAX 11/780.

We started the designs with an ISPS description† of RISC I and a block diagram similar to Figure 1. The logic, circuits, and initial layouts were designed on paper, and then entered into Caesar. As the designers became more comfortable with Caesar, they used it to do all layout. Caesar converts the graphical description into CIF (the format needed for fabrication submission, as well as for some of our tools, like CIFPLOT and DRC). The information necessary to run STAT, MOSSIM, and POWEST was extracted from this same CIF by MEXTRA. After the bottom level modules were designed, we used SLANG to completely describe the chip at a mixture of functional and logical levels. We then ran RISC diagnostic programs on this description to uncover errors. The SLANG description was also used to specify many of the remaining connections in the chip and to drive the PLA tools to automatically produce the PLA's for RISC. Howard Landman acted as a "roving critic", scanning for errors that were overlooked by the design tools.

† The ISPS description was not very useful, because the ISPS simulator does not run on the VAX (the machine we use to do our design); also, while useful for architecture descriptions, it is awkward for describing implementations.

The final step was to use SLANG and MOSSIM to compare the original description with the final masks. SLANG was used to interactively compare, every half clock phase, the values of hundreds of nodes in both the SLANG description simulation and the MOSSIM simulation. We ran about a dozen diagnostic programs and uncovered several errors.

The submitted Gold design successfully passed the checks provided by all of the tools.

5. CONTROL AREA, DESIGN TIME, AND REGULARITY

Table 3 compares several design metrics for RISC to those of some commercial microprocessors. The information for the Z8000 and MC68000 comes from[1] ; the information on the 432 comes from [3, 4], and [10]. The data provides quantitative support to some of the points made earlier.

The Gold chip is larger than most University (Mead-Conway) projects, but still comparable to the latest microprocessors. It is about 6 % larger than the 43203; on the other hand, the Blue chip may be 25 % smaller than the 43203. The most significant difference is that the simple instruction set of RISC I leads to a much smaller control section than that of the comparison processors; control occupies less than 10 % as compared to a more typical value of around 50 % for the other chips. When comparing absolute areas one finds that the smallest control section is still a factor of 5 larger than RISC's.

The smaller control section leaves correspondingly more area for the structured data path and, in particular, for the very regular and compact register file. Besides making effective use of the silicon area, this also increases the overall regularity of the chip. Lattin[4] defined a "regularity factor" as the *total* number of devices on the chip (excluding ROM's), divided by the number of *drawn* transistors. The table entries for the number of devices in the Z8000 and MC68000 are estimates, but the entries for the 432 and RISC are actual measurements. By this measure, RISC is 2 to 5 times more "regular".

The increased chip regularity was a key factor leading to a short design cycle. The elapsed time was considerably shorter for RISC; also, the effort spent (man months) was about a factor of five less. Control was by far the most time consuming part of the design and layout. Since the control section was reduced from half the chip to less than 10%, a significant reduction in design effort resulted.

Another factor for the reduced design cycle is our integrated design environment. All of the various tools, which performed graphic editing, checkplotting, design rule checking, layout rule checking, architectural simulation, and switch-level simulation, were at the fingertips of the designers and ran on the same machine. This reduced the overhead associated with the usage of such tools. Special credit is deserved by Caesar, the Manhattan-based graphic layout editor. Caesar was developed by Ousterhout[6] in close interaction with the chip designers. Bugs were corrected and designer wishes implemented, sometimes overnight. This responsiveness has produced a superb tool that is well-liked by the users since it dramatically improves their layout productivity.

Other factors affecting layout time were the use of the simplified Mead-Conway design rules, the restriction to Manhattan features, and (for the Gold design) the concentration on correct function rather than highest performance (SPICE simulations were used only in some of the most critical paths; however, extensive switch-level simulation was performed). Certainly, another portion of the reduced design time is due to the the fact that, as academics, we only had to deal with a set of rather loose, self-imposed constraints. For example, our master clock is generated off-chip, and the Gold chip has a rather poor external bus interface. The selection of op-codes was kept flexible to the very end; they were

Table 3.
Design metrics for Z8000, MC68000, iAPX-432, and RISC I.

	Zilog Z8000	Motorola 68000	Intel iAPX-432			RISC I	
			43201	43202	43203	Gold	Blue
Total Devices	17.5k	68k	110k	49k	60k	44k	37k+?
Total - ROM	17.5k	37k	44k	49k	44k	44k	?
Drawn Devices	3.5k	3.0k	5.6k	6.5k	5.7k	2.0k, 1.6k[a]	x+?
Regularization factor	5.0	12.1	7.9	5.2	7.7	21.7, 27.5[a]	?
Size of chip							
(Dimension in mils)	395x251	346x261	318x323	366x313	366x396	406x305	~400x~215
(Area in sq. mils)	99k	90k	103k	115k	117k	124k	~86k
Metal pitch (microns)	12	11	11	11	11	12	12
Size of Control[b]							
(Area in sq. mils)	37k	42k	67k	45k	47k	7k	~7k
Percent Control	53 %	62 %	65 %	39 %	40 %	6 %	~8 %
Elapsed Time to first silicon (months)	30	30	33[c]	33[c]	21[c]	17[d]	17+?[d]
Design Time (man months)	60	100	170[c]	170[c]	130[c]	30/2[e]	(30+?)/2[e]
Layout Time (man months)	70	70	90	100	50	12	5+?

[a] There are two ways to count drawn transistors; the pessimistic approach counts every transistor in a cell even if it derived from simple modifications to a basic cell. The optimistic approach only counts the transistors that were changed. For the Gold chip the difference is the transistor count for the register decoders. The optimistic count saves 433 drawn transistors thereby increasing the regularity factor.

[b] We estimated these sizes from the photomicrographs of the commercial chips.

[c] Data provided by Rattner of Intel [10]

[d] We counted elapsed time from the beginning of the first class to the end of the last class plus three months which should be the time for fabrication (we hope).

[e] Since the designers also did layout this is a somewhat fuzzy distinction. All work before 1/1/81 is considered design and we have included circuit design as part of layout.

modified the day before the design was submitted for mask generation!. An industrial product can rarely afford such luxury.

6. FINAL COMMENTS

The RISC Project has had a synergistic effect on research at Berkeley in architecture, VLSI, and CAD. Often, useful tools were created in response to specific needs. For example, a special extractor for Manhattan geometry was formulated since the older extractor, able to handle general geometry, would have taken too long to find all 44000 transistors of the Gold chip. As a result of this synergism our design environment has experienced a dramatic improvement within the last six months.

The gains in performance of the design tools by their restriction to Manhattan designs were well worth their inconveniences in layout; we can find only small areas in each chip where non-Manhattan geometry could save space.

While we realize that more work is necessary to turn RISC into a full-fledged microcomputer, we also believe that the most difficult and time consuming portion of that task has been completed. These results can be duplicated by industry; reduction in elapsed design time, reduction in manpower, and high performance are available for those who are willing to take calculated risks.

7. ACKNOWLEDGEMENTS

The RISC project was aided by several people at Berkeley and other places. We would like to thank them all, but give special thanks to a few:

John Ousterhout created *Caesar*, the main interface of the designers. The reliability and quality of this graphics editor and his responsiveness to our needs is a major reason for our reduced design time. **Richard Newton** allowed us to use his graduate class to resolve issues related to RISC. We also want to thank others in the Berkeley community for the use of their tools: **Bob Cmelik, Sheng Fang, Richard Newton,** and **Donald Pederson**. We would especially like to thank the people in the ARPA-VLSI community that shared their tools: **Randy Bryant** and **Chris Terman** for the switch-level simulator *MOSSIM* and **Clark Baker** for the layout rule checker *DRC*. In addition to these tools from MIT, from Stanford we received *STAT*, a static electrical rules checker created by **Forest Baskett**. We gratefully acknowledge helpful discussions with **Osamu Tomisawa** in the areas of MOS circuit design and processing. **Jim Beck, Bob Cmelik,** and **Robert Hyerle** provided valuable information and ideas on testing the chip. We would also like to thank the visitors from industry who gave us valuable suggestions on our designs: **Les Credele** from Motorola, **Dick Lyon** from Xerox PARC, and **Peter Stoll** from Intel. Thanks go to **Bob Fabry, Richard Fateman, Bill Joy,** and **Bob Kridle** for providing the computing environment necessary to complete our designs. We would like to thank **Danny Cohen** and **Lee Richardson** of the *MOSIS* group (U. of Southern California - Information Sciences Institute), and **Alan Bell** and **Al Pateth** of Xerox Palo Alto Research Center, for fabricating our chips. Finally, we would like to thank **Duane Adams** and **DARPA** for providing the resources that allow universities to attempt high-risk projects.

This research was sponsored by the Defense Advance Research Projects Agency (DoD), ARPA Order No. 3803, and monitored by Naval Electronic System Command under Contract No. N00039-78-G-0013-0004.

8. REFERENCES

1. Frank, E.H. and Sproull, R.F., "An Approach to Debugging Custom Integrated Circuits," *Carnegie-Mellon Computer Science Research Review 1979-80*, pp. 21-36 (July, 1981).

2. Joy, W.N., Fabry, R.S., and Sklower, K., *Seventh Edition, Virtual VAX-11 Version*, Computer Science Division, Dept of EECS, U. C. Berkeley June, 1981.

3. Lattin, W.W., Bayliss, J.A., Budde, D.L., Colley, S.R., Cox, G.W., Goodman, A.L., Rattner, J.R., Richardson, W.S., and Swanson, R.C., "A 32b VLSI Micromainframe Computer System," *Proc. IEEE International Solid-State Circuits Conference*, pp. 110-111 (February, 1981).

4. Lattin, W.W., Bayliss, J.A., Budde, D.L., Rattner, J.R., and Richardson, W.S., "A Methodology for VLSI Chip Design," *Lambda*, pp. 34-44 (Second Quarter, 1981).

5. Newton, A.R., Pederson, D.O., Sangiovanni-Vincentelli, A.L., and Séquin, C.H., "Design Aids for VLSI: The Berkeley Perspective," *IEEE Transactions on Circuits and Systems*, (July 1981).

6. Ousterhout, J., "Caesar: An Interactive Editor for VLSI Circuits," *VLSI Design* II(4)(Fourth Quarter, 1981). to appear

7. Patterson, D.A. and Ditzel, D.R., "The Case for the Reduced Instruction Set Computer," *Computer Architecture News* 8(6) pp. 25-33 (15 October 1980).

8. Patterson, D.A. and Séquin, C.H., "Design Considerations for Single-Chip Computers of the Future," *IEEE Transactions on Computers* C-29(2) pp. 108-116 (February 1980). Joint Special Issue on Microprocessors and Microcomputers.

9. Patterson, D.A. and Séquin, C.H., "RISC I: A Reduced Instruction Set VLSI Computer," *Proc. Eighth International Symposium on Computer Architecture*, pp. 443-457, (May 1981).

10. Rattner, J.R., *Private Communication*, August 1981