Simplified Microprocessor Design for VLSI Control Applications

K. Cameron

NASA Space Engineering Research Center for VLSI System Design
University of Idaho, Moscow, Idaho 83843
Phone: 208-885-6500 Fax: 208-885-7579

Abstract- **A design technique for microprocessors combining the simplicity of RISCs with the richer instruction sets of CISCs is presented. They utilize the pipelined instruction decode and datapaths common to RISCs. Instruction invariant data processing sequences which transparently support complex addressing modes permit the formulation of simple control circuitry. Compact implementations are possible since neither complicated controllers nor large register sets are required.**

1 Introduction

The design of microprocessors has evolved considerably since the introduction of the first microprocessor in 1971 [3]. Traditional microprocessors are extremely complicated machines that support hundreds of instructions and a dozen or so addressing modes. The dominance of such complex instruction set machines (CISC) has recently been challenged by much simpler processors which support only the most commonly used instructions. These processors are known as reduced instruction set computers (RISC) [8]. Traditionally, RISC processors:

- Support a small (reduced) instruction set of simple instructions which represent the most commonly used operations,
- Process instructions at the rate of one instruction per system clock cycle,
- Have a large on board register file for instruction or data cache.

The SPARC architecture is a scalable family of traditional RISC processors which was developed commercially. The architecture is deeply pipelined and depends heavily upon the compiler to efficiently map the register set and avoid forbidden instruction sequences [1]. Recently, the meaning of the term RISC processor has become quite blurred [6]. The IBM System/6000 purports to be a RISC implementation, but supports 184 instructions [5], which is a considerably larger instruction set than that of the 68020 microprocessor [7], which is generally considered to be a CISC machine.

The design methodology described here is targeted for applications which must be implemented in a small amount of circuitry (i.e. as a cell on a larger integrated circuit), while retaining medium to high levels of performance. The approach taken is to implement a system which adheres to most, but not all, of the design concepts of a traditional RISC machine. Key points of the design approach investigated are listed below. They will each be described in more detail later.

- The processor supports a very small (reduced) instruction set. Only vital or frequently used operations are supported directly.
- The instruction set is orthogonally partitioned. As nearly as possible, bit fields in instructions mean the same things for all instructions. All addressing modes are supported in the same manner for all instructions.
- All instructions are processed using invariant execution sequences. This means that information flows through the datapath in precisely the same manner for all instructions.
- Both the datapath and the associated controller are deeply pipelined. The use of invariant execution sequences permits the construction of very deep, yet simple processing pipelines.
- Only a small internal register set is supported. The processor registers are memory mapped, allowing them to be accessed and updated with general memory reference instructions.
- The support of relatively complex addressing modes is important if the internal register set is small. Implemented consistently across the entire instruction set, they add little to the overall complexity of the machine.

Though a specific processor was implemented, the design methodology followed may be used to implement a large number of different RISC-like processors, each with different size-performance trade-offs.

2 Execution Cycle Pipeline

The data flow strategy of the microprocessor is the first item which must be designed. This includes data flow to and from memory as well as through the datapath of the processor itself. The performance required of the processor drives the choices made at this point. Different cost–performance ratios can be achieved through the use of different data flow strategies. A few of the possible tradeoffs are listed below:

- Processor word size?
- Separate address and data busses used to access memory?
- Separate instruction fetch and program data stores?
- Separate address generation and data processing units?
- Multiple data processing units?
- Pipeline depth?
- Data/instruction cache?

Ram Access	Fetch1	Addr0	Fetch2	Addr1	Fetch3	Addr2	Fetch4	Addr3
Decode		Inst1		Inst2		Inst3		Inst4
Alu		Addr1	Op0	Addr2	Op1	Addr3	Op2	Addr4
Adder			Addr1	Op1	Addr2	Op2	Addr3	Op3

Figure 1: μP Execution Sequence

- Number of internal data/address registers?
- Maximum number of instructions?

Since the design implemented was to have moderate performance yet be compact, it was decided to build a processor with a 16 bit word, separate address and memory busses, shared data and instruction stores, combined address generation and data processing units, a deep pipeline, no cache, and a small set (4) of general registers. It was further decided that processor registers would be memory mapped, so separate instructions would not have to be provided to access either the processor registers or the registers associated with the accompanying IO subsystem.

Though shallow pipelines such as was used with RISC II [11] are relatively simple to design, it was decided from a performance stand-point that the machine should be deeply pipelined. A deep pipeline permits the construction of a high-throughput processor, since each stage of the pipe can operate independently on different portions of the problem at the same time. Deep pipelines, however, have the undesirable characteristic that any irregularity in the processing sequence for different instructions can lead to either the need for extremely complicated locking circuitry [5,1] or else the definition of a large number of forbidden instruction sequences to prevent data collisions. It was, therefore, decided that all instructions should share the same (though perhaps, a truncated) processing sequence. The memory execution sequence finally decided upon is shown in Figure 1. Several key points of this processing sequence are:

- Each RAM access is pipelined two clock cycles deep. This greatly eases all timing paths associated with RAM accesses.
- Data associated with an instruction "wraps" through the ALU/Adder twice. Once to calculate the associated address and once to process data. This strategy keeps the datapath completely utilized at all times.
- Data processed during one instruction is available for subsequent processing on the very next instruction.
- One instruction is executed every two clock cycles.

Mode	Invoked	Description
Direct	$s = 0$	Effective Address is part of instruction.
Indexed	$s = 1 \;\; Offset \neq 0$	Effective Address is contents of referenced stack pointer plus signed offset.
Stack	$s = 1 \;\; Offset = 0$	If instruction implies a read, the referenced pointer is pre-incremented. The Effective Address is the new stack pointer contents. If the instruction implies a write, the Effective Address is the contents of the references stack pointer. The stack pointer is post-decremented.

Figure 2: μP Addressing Modes

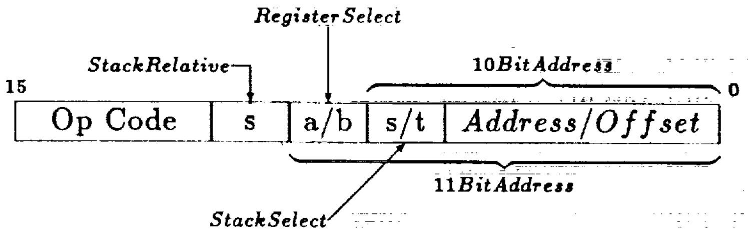

Figure 3: μP Instruction Format

3 Instruction Types

Once data flow strategy has been determined, the instruction set and addressing modes of the processor must be selected. Here, a wide variety of possibilities presents itself. Since the processor implemented is intended for an interrupt driven environment, it was decided that the machine should be stack oriented and provide good support for stack based operations. The addressing modes summarized in Figure 2 were finally decided upon. Direct referencing of memory locations with a pointer contained in the instruction itself provides simple access of memory mapped registers, global variables and targets for normal jump and jump subroutine instructions. The indexed with offset mode provides support for jump tables, arrays, stack oriented local variable access, and subroutine argument passage. The auto-decrement and increment modes support implied push and pop operations as a part of any instruction, ease the placing of arguments on the stack for passage to subroutines, and allow the return from subroutine instruction to be implemented as a special case of the jump instruction.

Figure 4 summarizes the instructions set which was selected. Each instruction can be operated in any addressing mode. The actual instruction format is shown in Figure 3.

Op Code	Mnemonic	Register	Description
0100	ld	a/b	Load Register
1111	st	a/b	Store Register
1000	jsr	—	Jump to Subroutine
0000	jmp	—	Absolute Jump
0010	and	a/b	Bitwise And
0011	or	a/b	Bitwise Or
1100	add	a/b	Addition
1110	sub	a/b	Subtraction
0110	not	a/b	Bitwise Complement
0101	xor	a/b	Bitwise Exclusive Or
1101	cmp	a/b	Skip next instr if not equal
1010	tst	a/b	Bitwise And, then skip next if 0
0111	shl	a/b	Shift Left
0001	shr	a/b	Shift Right
1001	lds	s/t	Load Stack Register
1011	ien	—	Enable/Disable Interrupts

Figure 4: μP Instruction Set

Since the arithmetic unit already provides an adder and a zero detect circuit for the implementation of the base instruction set, virtually no additional hardware in the datapath was required to implement the addressing modes. If a hardware multiplication instruction had been included in the instruction set, it would have been possible to utilize it during address generation to provide very sophisticated support for array accessing.

The requirement that all instructions be implemented with the same processing sequence places severe restrictions on the type of conditional statements that can be provided, however. A test and skip next instruction pattern was selected since it fits the required schema and was possible to implement without disturbing the pipelined flow of instructions. No retry of instructions is necessary, since the results of the test are always known in time to abort the effects of any subsequent instruction.

4 Implementation

4.1 General

The processor was designed using structured logic design techniques in a custom environment. High operating speeds and compact layouts were achieved through the extensive use of pass-logic.

The use of an orthogonally partitioned instruction set and an instruction invariant processing sequence resulted in extremely small and simple control circuitry. Consequently, the speed of a machine cycle is limited only by delays in the datapath– not by propagation

Cycle1:	$PC \xrightarrow{p} AR$	$AR \xrightarrow{p} AR$	
Cycle2:	$RAM \xrightarrow{m} MO$	PC^{++}	
Cycle3:	$MO \xrightarrow{i;alu} Pipe2$	$SP^{(--)} \xrightarrow{r;alu} Pipe1$	$0 \xrightarrow{r;alu} Pipe1$
Cycle4:	$Pipe1 \xrightarrow{add;o;p} AR$	$Pipe2 \xrightarrow{add;o;p} AR$	$Pipe1/Pipe2 \xrightarrow{add;o} SP$
	$A/B/PC \xrightarrow{q} MI$	$SP^{(--)} \xrightarrow{p} AR$	
Cycle5:	$RAM \xrightarrow{m} MO$	$MI \xrightarrow{m} RAM$	
Cycle6:	$MO \xrightarrow{i;alu} Pipe2$	$A/B \xrightarrow{r;alu} Pipe1$	$0 \xrightarrow{r;alu} Pipe1$
Cycle7:	$Pipe1 \xrightarrow{add;o} A/B$	$Pipe2 \xrightarrow{add;o} A/B$	$Pipe1/Pipe2 \xrightarrow{add;o}$

Figure 5: μP Register Transfer Sequences

delays in the controller.

The processor itself is a simple design. Approximately three man months were required to design the circuit and verify its logical correctness through extensive logic simulations. During this time a software model of the processor was also written to aid logic verification and a macro-assembler was written for software development. Four man months were required to implement the layout and verify its correctness.

The processor was implemented in a 1.6μm CMOS process and subsequently shrunk to a 1.0μm process due to size considerations. It runs at a clock frequency of 28MHz under worst case processing assumptions, 140deg C junction temperature, and 4.1V internal supplies. The processor was completely functional on first silicon. Under typical conditions, it should run at nearly 60MHz, which corresponds to an instruction rate of 14MIPS worst case and 30MIPS typical. Currently, the limiting speed path is associated with the zero detect circuit. A redesign of this circuit would likely result in yet higher system performance.

4.2 Control

The upper bits the data bus are fed into the control section where they are pipelined parallel to the data passing through the datapath. The instruction decode and control of the datapath is simple, since both control and data are pipelined in an equivalent manner. The individual control lines to the datapath are decoded directly from the pipelined instructions. The logic of the control section fits on one C sized sheet of logic. It consists of four stages of pipeline registers, 61 NAND gates (most of which are 2 to 4 input gates), 10 NOR gates, and various inverter/buffers. It contains no state-machines except those required for interrupts and memory cycle stealing by the IO subsystem— and these are exclusively single bit state-machines.

4.3 Data Path

The datapath consists of a register stack and a pipelined Adder and ALU. Figure 6 is a signal flow diagram of the datapath. The M Bus is used for all memory mapped data

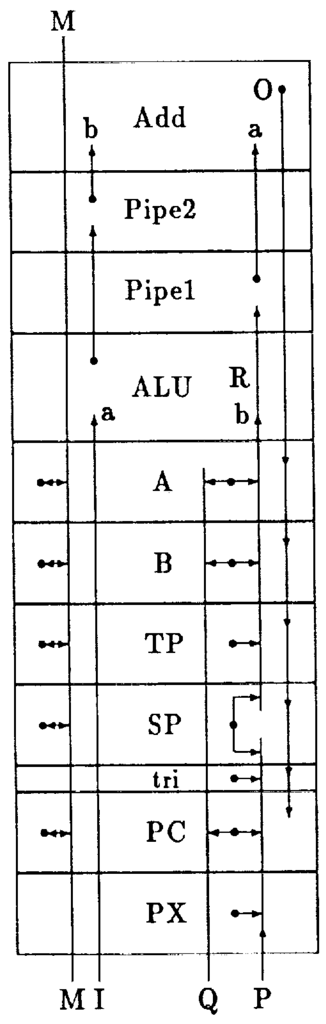

Figure 6: μP Register Stack

transfers. The P Bus drives the Address bus of the RAM through a clocked register, AR, located in the pads. The I bus is the data input bus from the memory, and the Q bus is the data bus to the RAM. The I and Q bus are combined into a single bi-directional bus at the chip pads through the MI and MO registers. (MI receives data from RAM during a read and MO outputs data to RAM during a write.) The datapath operates as follows. The instruction fetch address is driven from either the program counter or the secondary program register onto the P (address) bus. Two clock cycles later, the instruction arrives on the I bus, where it is fed through the ALU. At this time the Op Code portion of the instruction is stripped off and the remaining bits are used to form either an absolute address or and offset for the stack relative mode of operation. The results are clocked into the Pipe registers. Next clock cycle, this address/offset is either passed unaffected through the ADDER (absolute addressing mode) or added to the contents of the selected address register (SP or TP) (indirect addressing mode), and the results are driven onto the P (address) bus through a tri-state driver. If the instruction implies a write to memory, the appropriate data is driven onto the Q bus from either A,B or PC. If the instruction implies a read the requested data enters the datapath via the I bus two clock cycles later and is processed by the ALU and ADDER in succession, at which time the results are loaded into the appropriate register. An RTL description of the data transfers comprising the data processing sequences utilized to implement the entire instruction set is shown in Figure 5. It should be noted again that though this processing sequence is seven clock cycles deep, processing of a new instruction starts every other clock cycle.

The ALU and adder are both implemented using pass logic. The ALU consists of a single cell replicated 16 time, each of which consists of only 23 n-channel pass gates and 9 inverter/buffers. The ALU performs all bitwise operations and provides a zero detect function which is used in the conditional skip instructions, as well as the detection of the auto increment/decrement addressing modes. The OpCode (figure 4) bit patterns were selected such that the upper bits of the instructions themselves become the control lines for the ALU with minimal remapping.

The configuration selected to implement the ADDER is a modified transmission gate conditional sum scheme [10]. The configuration is small, regular, and very fast.

4.4 IO Subsystem

Though not a primary topic here, it should be mentioned that a complete IO subsystem was implemented and integrated with the microprocessor described here. It consisted of a DMA subsystem which was responsible for the bulk transfer of data around the chip, two serial ports for low speed data transmission and acquisition, a parallel port for the transfer of data to and from an external microprocessor, as well as a prioritized interrupt/event passage system.

5 Conclusion

Present day integrated circuit fabrication processes support levels of integration adequate for the construction of on-board microprocessor based controllers which occupy only a small portion of the available circuit area. Such processors can be readily designed for different cost-performance tradeoffs, as required for specific applications. The outlay of engineering time need not be excessive and the use of high-level languages for code development makes the underlying instruction set transparent to the firmware developer, and eases code migration, development and support.

References

[1] A. Agrawal et.al., "The Scalable Processor Architecture (SPARC)," COMPCON '88 Proceedings, 1988, pp. 278-283.

[2] H. Bakoglu, T. Whiteside, "RISC System/6000 Processor Architecture," IBM RISC System/6000 Technology, SA23-2619, IBM, Austin TX, 1990, pp. 8-15.

[3] D. Curtin, L. Porter, *Microcomputers: Practices and Procedures*, Prentice-Hall, 1986.

[4] G. Grohoski, J. Kahle, L. Thatcher, C. Moore, "Branch and Fixed-Point Instruction Execution Units," IBM RISC System/6000 Technology, SA23-2619, IBM, Austin TX, 1990, pp. 24-32.

[5] P. Hester, "RISC System/6000 Hardware Background and Philosophies," IBM RISC System/6000 Technology, SA23-2619, IBM, Austin TX, 1990, pp. 2-7.

[6] J. McLeod, "Tough Choices Ahead in Microprocessors," *Electronics*, May 1989, pp. 70-78.

[7] *MC68020 32-Bit Microprocessor User's Manual*, 2nd Edition, ISBN 0-13-566860-3, Prentice-Hall, Englewood Cliffs, NJ, 1985, p. 1-6.

[8] D. Patterson, C. Sequin, "A VLSI RISC," IEEE Computer, vol. 15, No. 9, Sep 1982, pp. 8-12.

[9] C. Rowen et.al., "RISC VLSI Design for System Level Performance," VLSI Systems Design, March 1986, pp. 81-88.

[10] A. Rothermel, et al., "Realization of Transmission-Gate Conditional-Sum (TGCS) Adders with Low Latency Time," IEEE JSSC, Vol. 24, June 1989, pp. 558-561.

[11] R. Sherburne, M. Katevenis, D. Patterson, C. Sequin, "A 32b Microprocessor with a Large Register File," Digest of IEEE International Solid-State Circuits Conference, Feb 1984, pp. 168-169.