Future Generation Computer Systems 7 (1991/92) 303–309
North-Holland

SPARC: A scalable processor architecture

Anant Agrawal and Robert B. Garner

Sun Microsystems, 2550 Garcia Avenue, Mountain View, CA 94043, USA

Abstract

Agrawal, A. and R.B. Garner, SPARC: A scalable processor architecture, Future Generation Computer Systems 7 (1991/92) 303–309.

SPARC defines a general purpose 32-bit scalable processor architecture. The simple yet efficient nature of the architecture allows cost effective and high-performance implementations across a range of technologies. Since its first implementation done in Fujitsu's C20K gate array, a number of implementations have been announced in various technologies including bipolar ECL. All these designs implement the same instruction set. Thus an application program behaves identically and produces the same results on all SPARC platforms executing the operating systems that support the architecture.

Keywords. RISC, SPARC; processor; architecture; ISA; instruction set; register windows; workstation; performance; NC Berkeley; Sun Microsystems; Sun4.

1. Introduction

SPARC™ logically consists of a 32-bit integer unit, an IEEE-standard floating-point unit and a user-defined co-processor unit. Each unit has its own set of registers. This enables maximum concurrency between these units. The architecture assumes a linear, 32-bit virtual address space for user-application programs.

One of the design goals for SPARC was to define a very simple yet efficient architecture that could be implemented cost effectively in various technologies, where some of them could be faster and possibly less dense. Another goal was support for high level languages and keeping compilers relatively simple.

This paper introduces the SPARC architecture. Also covered is the suitability of the architecture for realtime applications. A complete architectural specification is available in ref. [14]. Papers have been written that cover the architecture [5], compilers [8], SunOS on SPARC [7], and the Fujitsu [9,12] and Cypress [10] implementations. Ref. [1] is about some of the design considerations for the bipolar ECL implementation of SPARC. An introduction to RISCs is found in ref. [11].

2. Integer unit

SPARC defines 55 basic integer instructions and their variations. Instructions include a comprehensive set of logical, arithmetic, control transfer, memory reference and multiprocessor instructions. Support for AI languages is provided through tagged arithmetic instructions.

2.1. Registers

SPARC is a register intensive architecture where a large bank of registers is divided into sets of overlapping registers known as *windows* [6]. The architecture defines up to 32 windows. The actual number may vary across implementations. Thus, the IU may contain from 40 to 520 registers [1]. Each window consists of 32 registers which are divided into 8 *global* registers (same for all windows), 8 *ins*, 8 *locals* (unique to each window), and 8 *outs* (as shown in *Fig. 1*). Adjacent register windows share eight registers (*outs–ins*). Overlapped windows provide an efficient way to pass

[1] A minimal, 40-register, two-window implementation comprises 8 *ins*, 8 *locals*, 8 *outs*, 8 *globals*, and 8 trap handler *locals*.

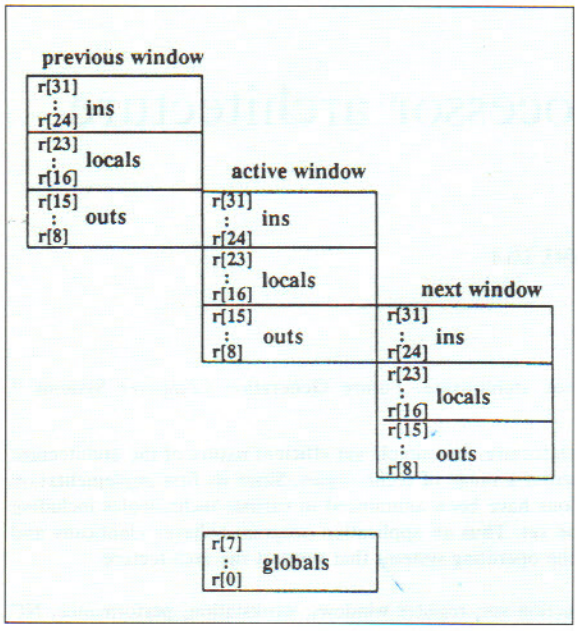


Fig. 1. Three overlapping windows and globals.

parameters during procedure calls and returns. The compiler paper [8] explains how windowed registers can be used.

The active window is identified by the current window pointer (CWP), a 5-bit pointer, within the processor state register. Decrementing the CWP at procedure entry causes the next window to become active and incrementing the CWP at procedure exit causes the previous window to become active. An IU state register, the window invalid mask (WIM), is used to tag a window (or sets of windows). An overflow or underflow trap occurs if, due to an operation that changes the CWP, it is about to point to a tagged window. To implement the usual LIFO stack of overlapping windows, one of the WIM bits is set to identify the boundary between the oldest and the newest window.

Register windows have several advantages over a fixed set of registers. Their principal advantage is a reduction in the number of load and store instructions required to execute a program. As a consequence, there is also a decrease in the number of data cache misses. The reduced number of loads and stores is also beneficial in implementations that have multi-cycle load or store instructions and in tightly coupled multiprocessors.

Register windows also work well in incremental compilation environments such as LISP and in object-oriented programming environments such as Smalltalk, where interprocedural register allocation is impractical. Even though these exploratory programming languages benefit from register windows, SPARC does *not* preclude interprocedural register allocation optimizations since the subroutine call and return instructions are distinct from the instructions that advance and retract the window pointer.

In addition to the window registers there are a number of architecturally defined registers in the integer unit; the PSR which holds the integer unit's processor state, that includes the user/supervisor bit, the integer condition codes, the current window pointer (CWP), the FP/CP disable bits, the 4-bit processor interrupt level PIL and an 8 bit version/implementation number; the window invalid mask (WIM); the trap base register (TBR); the program counters (PC and NPC) and the multiply step register (Y). These are described in detail in ref. [14].

2.2. *Instructions*

All the SPARC instructions are 32-bits wide and are defined by one of the three formats

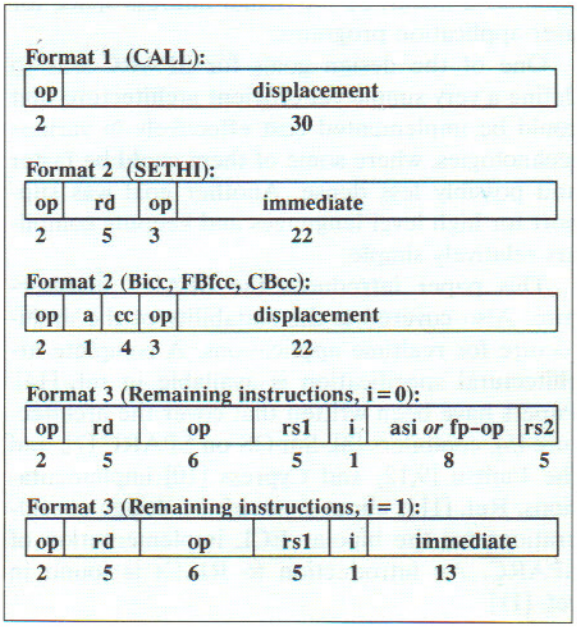


Fig. 2. SPARC instruction formats.

illustrated in *Fig. 2*. Special care was taken in encoding instructions to enable the fastest possible implementations.

Format 1 defines a PC-relative CALL instruction with a 30-bit word displacement. Thus a call or an unconditional branch can be made to any arbitrary location in the address space with a single instruction.

SETHI and branch instructions use format 2. This format defines a 22-bit immediate field. For PC-relative branches it provides ± 8 Mbyte of displacement. SETHI loads the immediate value into the high 22 bits of the destination IU register and clears its low 10 bits. SETHI, in conjunction with a format 3 instruction, can be used to create 32-bit constants.

Format 3 encodes the remaining instructions including floating point and co-processor instructions. It specifies a destination register and either two source registers or a source register and a 13-bit sign extended immediate field. Eight bits of the immediate field are used as an opcode extension field for specifying floating-point/co-processor instructions and, as an "address space identifier" for the load/store instructions.

2.2.1. Memory reference instructions

Memory can be accessed only through load/store instructions. For all such instructions, including floating-point and co-processor load/stores, the IU generates the memory address and the IU, FPU or co-processor sources or sinks the data.

All memory reference instructions use format 3, and support both "$\text{reg}_1 + \text{reg}_2$" and "reg + signed_13-bit_constant" addressing modes. Register indirect and absolute addressing modes can be emulated using g0. Load/store instructions support signed/unsigned byte, half-word, word and double word transfers. If the data is not aligned at the proper boundary, the instruction traps. Big-endian or the IBM 370 compatible byte-ordering is supported. Byte 0 is the most significant byte in a datum.

For all instruction fetches and normal data accesses the IU provides a 32-bit virtual address and an 8-bit address identifier (ASI). Data fetches could be either *normal* or *alternate*. For all instructions and *normal* data fetches ASIs indicate a user/supervisor and data/instruction reference. This can be used to provide a protection mechanism in a system's memory management units.

Alternate instructions are privileged and can be executed only in supervisor mode. Their format is restricted to "$\text{reg}_1 + \text{reg}_2$". They use eight ASI bits to specify either the user instruction or user data spaces, or up to 252 other system-dependent, 32-bit address spaces. The SPARC architecture defines only user/supervisor, instruction/data spaces; the remainder can be defined by the system architecture.

Unlike many other RISC architectures, SPARC does not have "delayed loads". An instruction immediately following a load instruction may use the load data. This simplifies the job of scheduling instructions by compilers. Depending on implementation this case may cause the sequence to take an additional cycle to complete the operation.

2.2.2. Multiprocessor instructions

SWAP and load–store unsigned byte (LDSTUB) instructions provide support for tightly coupled multiprocessors. SWAP exchanges the contents of an IU register with a word from memory. It can be used in conjunction with a memory-mapped co-processor to implement synchronizing instructions, such as the non-blocking "fetch and add" instruction. LDSTUB reads a byte from memory into an IU register and then rewrites the same byte in memory to all ones. It can be used for blocking synchronization schemes, such as semaphores [4]. Both the instructions are atomic.

2.2.3. Arithmetic / logical instructions

These format 3 integer instructions perform either a logical or an arithmetic operation on two operands and optionally write the result into a destination register. Arithmetic instructions have two types: ones that update the integer condition codes and ones that do not. There are four condition codes, negative (N), zero (Z), overflow (V) and carry (C). They are stored in the processor state register.

The "multiply step" instructions (MULScc) are used to generate the 64-bit product of two signed or unsigned words in multiple cycles. Though (MULScc) processes the multiplier one bit at a time, as mentioned in the compiler paper [8], higher-level language multiplications execute in an average of six cycles.

2.2.4. Tagged instructions

These instructions provide support for languages that can benefit from operand tags, such as LISP and Smalltalk. They assume 30-bit left justified signed integers and use the least significant two bits of a word as a tag. The "tagged add/subtract" instructions (TADDcc, TSUBcc) set the overflow condition code bit if either of the operands has a nonzero tag (or if a normal arithmetic overflow occurs). Normally, a tagged add/subtract is followed by a conditional branch instruction, which, if the overflow bit has been set, transfers control to code that further deciphers the operand types. Two variants, TADDccTV and TSUBccTV, trap if the overflow bit has been set and can be used to detect operand type errors.

2.2.5. Special instructions

These instructions are used to read and write architecturally defined registers. Some of them are privileged and can be executed only in the supervisor mode. SAVE and RESTORE instructions are used to decrement or increment the current window pointer. They trap if the adjustment would cause a window overflow or underflow. They also operate like an ordinary ADD instruction and thus can also be used to atomically adjust a program stack pointer.

2.2.6. Control transfer instructions

These instructions consist of call, branch, jump and link and trap on condition code instructions. For efficient execution of these instructions, SPARC uses the concept of delayed branches. For most of these instructions the instruction that follows the control transfer instruction is executed before program control is transferred to the target instruction.

Compilers try to move a useful instruction from a location before the branch into the delayed slot. When this is not possible, a NOP is generally placed in the delay slot. However, SPARC conditional branches have a special "annul" bit. If the annul bit is set and the conditional branch is not taken, the delay instruction is *not* executed. This feature allows compilers to move an instruction from the target, or move an instruction from one arm of an IF–THEN–ELSE statement into the other. By use of the annul bit, compiled code contains less than 5% NOPs.

Traditional non-delayed branches can be emulated using the "branch always" (BA) instruction. If a BA with the annul bit set is executed, its delay instruction is *not* executed. It can also be used to efficiently emulate unimplemented instructions if, at runtime, the unimplemented instruction is replaced with an annulling BA whose target is the emulation code.

The "trap on condition code" (Ticc) instructions do not have a delay slot and they conditionally transfer control to one of 128 software trap locations. Ticc's are used for kernel calls and compiler run-time checking.

3. Floating-point unit

SPARC defines fourteen basic floating-point instructions. IEEE single, double and extended precision data types are supported. The FPU has thirty-two 32-bit-wide registers. Double-precision values occupy an even–odd pair and extended-precision values occupy an aligned group of four registers. Data cannot be transferred directly from IU registers to FPU registers, it has to be done through the memory. The instruction set defines double-word (64-bit) floating-point loads and stores to boost double-precision performance. Also, in order to decrease context switch time, the FPU can be disabled so that its registers need not be saved when switching from a process that does not use floating-point.

SPARC allows floating-point operations, such as multiply and add, to execute concurrently with each other, with floating-point loads and stores, and with integer instructions. This concurrency is hidden from the programmer: a program generates the same results, including traps, as if all instructions were executed sequentially.

Because of this concurrency, the IU's program counters can advance beyond floating-point instructions in the instruction stream, before the floating-point instruction completes. There is a special group of registers, the floating-point queue (FQ), that records the floating-point instructions (and their addresses) that were pending completion at the time of a floating-point trap. The queue's head contains the unfinished instruction (and its address) that caused the floating-point trap.

Floating-point operations can also execute concurrently with cache misses. If a floating-point store attempts to write a result whose computation has not yet finished, the IU stalls until the floating-point operation is complete. A "store FSR" instruction also causes the FPU to wait for outstanding floating-point operations to finish.

The "floating-point operate" instructions (FPop) are specified via the 9-bit "opf" field of format 3 instructions. They compute a single, double, or extended-precision result that is a function of two source operands in FPU registers and write the result into FPU registers. The floating-point compare instructions write a 2-bit condition code in the FPU's floating-point status register (FSR) that can be tested by the "branch on floating-point condition codes" (FBfcc) instruction. There are instructions that convert between all formats, including integers.

In general, a user program sees a complete ANSI/IEEE 754-1985 implementation, even though the hardware may not implement every nuance of the standard, such as gradual underflow. Software emulates missing hardware functionality via FPU-generated traps.

4. Co-processor

As mentioned earlier, SPARC has instruction support for a single co-processor (in addition to the floating-point unit). The co-processor instructions mirror the floating-point instructions: load/store co-processor, "branch on co-processor condition codes", and "co-processor operate" (CPop). Co-processor operate instructions, can execute concurrently with integer instructions, and have not been defined as a part of the SPARC architecture.

5. Real time applications

The SPARC allows for fast trap handling, which is very useful in real time applications. The register windows in SPARC always provide eight free registers for trap handling, thus allowing for fast entry into trap handlers. When a trap or interrupt occurs, the CWP is decremented – as for a procedure call – making available to the trap handler six of the *local* registers of the next window. (Two of the *locals* are written with the IU's two program counters.) For a simple handler it takes about six cycles (in the first implementation) to enter the handler and another six to exit. [2] This assumes no cache misses. For cache misses this number would increase depending on the miss cost. If more registers than allocated for the handler are required, then a window save is necessary. This will increase the cost of interrupt handling. Note that on process switches only the active windows are saved, *not* the entire set of windows. [3]

The register windows can be managed in a variety of different ways for different applications. For applications that require rapid context switching, such as device controllers, the windows can be partitioned into non-overlapping pairs, with one pair allocated per process. At process switch time, pairs of windows could be switched providing each process with 24 private registers plus 8 *global* registers and a set of 8 registers for trap handling. With a large register file a number of these contexts can simultaneously reside in the processor and the WIM could be used to protect each process's registers from the other processes.

5.1. Traps and exceptions

Execution of a given instruction can raise several traps or exceptions. The source of a trap can be internal (synchronous) or external. In both cases the IU handles the trap in a similar manner. All traps raised during the execution of any instructions are deferred to the last stage of the pipeline, at that stage the highest priority trap is taken. Traps are vectored using the *trap base address register* (TBR) to point to the trap table. When a trap is taken, the current window pointer is decremented, further traps are disabled, the two program counters (PC and NPC) are automatically saved and the program continues at the trap vector location. External interrupts are given to the IU using 4-bit interrupt input signals. A non-zero value on these inputs is detected by the IU as an external interrupt request. This value is

[2] This assumes that the handler runs with traps disabled, there are no subroutine calls and only five of the local registers are required.

[3] The cost of saving or restoring a window is not large: on the Sun-4/200, it approximates the overhead of 7 cache misses.

compared with the processor interrupt level in the PSR and an interrupt is taken if the external interrupt request level is greater than the processor interrupt level. The highest level interrupt (level 15) is defined to be non-maskable, although all traps can be disabled via the enable trap ET-so bit in the PSR.

All external interrupts are ignored when traps are disabled. If a synchronous trap is detected while traps are disabled, the IU enters into an error mode and remains in that mode until it is reset by external logic. At reset, the IU is initialized and starts execution from address zero.

Floating-point exceptions, in general, occur asynchronously with respect to the IU pipeline since integer instructions are executed concurrently with the floating-point instructions and floating-point instructions take a variable number of cycles to complete their execution or generate an exception. However, in SPARC floating-point exceptions are taken *synchronously*. Floating point exceptions detected during the execution of an instruction are kept pending till another floating point instruction enters the IU pipeline. At that time the floating-point trap is taken (if it is the highest priority).

6. Conclusion

This paper summarized the SPARC architecture. It has been licensed to a number of different semiconductor companies which has resulted in various implementations at a variety of price–performance points. The simplicity of the architecture resulted in its being the first 32-bit general purpose architecture to be implemented in a gate array. Within a short time after that we saw its ECL implementation from BIT. With a number of companies working on different implementations we expect to see SPARC in both emerging and mature technologies.

Acknowledgements

Many people at Sun Microsystems contributed to the definition of the architecture, including Faye Briggs, Emil W. Brown, David Hough, Bill Joy, Steve Kleiman, Steven Muchnick, Masood Namjoo, Dave Patterson, Joan Pendelton, Richard Tuck, Dave Weaver, Dave Goldberg, Tom Lyon, Alex Wu, and John Gilmore. The gate-array IU (Fujtisu MB86900) was designed by Anant Agrawal and Masood Namjoo. Don Jackson designed the FPC (Fujitsu MB86910) with additional help from Rick Iwamoto and Larry Yang. Will Brown wrote an architectural and machine cycle simulator. Ed Kelly and Robert Garner designed the Sun-4/200 processor board. Wayne Rosing and K.G. Tan managed the architecture and the gate-array projects, Jim Slager managed the custom CMOS implementation and the author managed the bipolar ECL implementation.

UNIX is a trademark of AT&T Bell Laboratories. SPARC and Sun-4 are trademarks of Sun Microsystems, Inc. VAX is a trademark of Digital Equipment Corp.

References

[1] A. Agrawal, E.W. Brown, J. Petolino and J. Peterson, Design considerations for a bipolar implementation of SPARC, *IEEE COMPCON 88*.

[2] E. Brown, A. Agrawal et al., Implementing SPARC in ECL, *IEEE Micro* (Feb. 1990).

[3] N. Chu, L. Poltrack, J. Bartlett, J. Friedland and A. MacRae, *Sun Performance*, Sun Microsystems, Inc., Mountain View, CA.

[4] M. Dubois, C. Scheurich and F. Briggs, Synchronization, coherence and ordering of events in multiprocessors, to appear in *IEEE Com.*.

[5] R. Garner, A. Agrawal, F. Briggs, E.W. Brown, D. Hough, B. Joy, S. Kleiman, S. Muchnick, M. Namjoo, D. Patterson, J. Pendelton and R. Tuck, The scalable processor architecture (SPARC), *IEEE COMPCON 88*.

[6] M. Katevenis, Reduced instruction set computer architectures for VLSI, Ph.D. dissertation, Computer Science Div., Univ. of California, Berkeley, 1983. Also published by M.I.T. Press, Cambridge, MA.

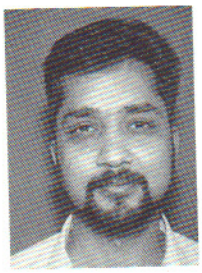
[7] S. Kleiman and D. Williams, SunOS on SPARC, *IEEE COMPCON 88*.

[8] S. Muchnick, C. Aoki, V. Ghodssi, M. Helft, M. Lee, R. Tuck, D. Weaver and A. Wu, Optimizing compilers for the SPARC architecture: an overview, *IEEE COMPCON 88*.

[9] M. Namjoo, A. Agrawal, D. Jackson, L. Quach, CMOS gate array implementation of the SPARC architecture, *IEEE COMPCON 88*.

[10] M. Namjoo, et al., CMOS custom implementation of the SPARC architecture, *IEEE COMPCON 88*.

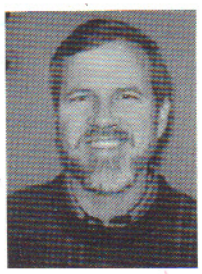
[11] D. Patterson, Reduced instruction set computers, CACM, vol. 28, no. 1, Jan. 1985.
[12] L. Quach and R. Chueh, CMOS gate array implementation of the SPARC architecture, *IEEE COMPCON 88*.
[13] M. Schafir and A. Nguyen, *Sun-4/200 Benchmarks*, Sun Microsystems, Inc., Mountain View, CA.
[14] *The SPARC™ Architecture Manual*, Sun Microsystems, Inc., Mountain View, CA. Also published by Fujitsu Microelectronics, Inc., 3320 Scott Blvd., Santa Clara, CA 95054.



Anant Agrawal, currently director of VLSI in SPARC group at Sun Microsystems Inc., was one of the developers of the SPARC architecture, and a principal engineer of the the first SPARC chip. Anant has managed the development of multiple SPARC chips at Sun including the ECL SPARC processor, and is currently managing two SPARC processor developments.

Prior to joining Sun Mr. Agrawal was at STC computer Research Corporation where he was the lead technical designer on the Floating Point Unit for an IBM-compatible mainframe.

Mr. Agrawal received a BSEE degree from M.S. University in Baroda, India in 1977 and an MSEE from Cornell University in 1979.



Robert Garner, currently a hardware manager at Sun Microsystems, was one of the principal architects of Sun's Scalable Processor Architecture, or 'SPARC', and a principal engineer of the first Sparc-based product, the Sun-4/200 Series. The Sun-4/200 was the first 10-MIPS RISC workstation in the market and won a 1987 Japan Nikkei Award for 'Creative Excellence in Products and Services'. Sparc was selected by Fortune magazine as a 'Product of the Year' in 1987. The Sparc architecture has also been licensed to several semiconductor and systems companies.

Prior to Sun, Mr Garner was a member of the research staff at the Xerox Palo Alto Research Center (PARC). Previously, in the Systems Development Division (SDD), he was a principal hardware engineer for Xerox's 8000 Series 'STAR' Information System, which was also marketed as the Xerox 1100 Series of programming workstations. Star was the first commercial workstation and integrated office system in the market and was selected by Fortune magazine as a 'Product of the Year' in 1981.

Mr. Garner received a BSE from Arizona State University in 1976 and an MSEE from Stanford University in 1977.