ARC Core Gives ASICs Programmability

London-Based Argonaut Develops Yet Another Synthesizable 32-Bit RISC Core

by Jim Turley

Argonaut Technologies, on a quest to design the most flexible 32-bit microprocessor core yet, has released its ARC (Argonaut RISC core) design, a synthesizable core for ASIC developers. Like many companies before it, Argonaut claims to have achieved the optimal balance between performance, small die size, and flexibility.

The definition of the ARC architecture is extremely loose and extensible. At its most basic, the CPU has a simple pipeline and only 16 basic opcodes. The core can be implemented in a few as 16,000 logic gates, nearly half of which are devoted to the register file. ARC's advantages come from its extensibility; licensees and users are encouraged to add registers, new instructions, and function units as they see fit.

The ARC design is delivered as a VHDL model rather than a physical macro. Although no chips have ever been fabricated, the company claims the basic ARC core can be synthesized into just 4 mm^2 in a 0.5-micron two-layer-metal CMOS process and would run at 50–70 MHz.

Not Quite Like Other Microprocessor Cores

Realistically, Argonaut doesn't expect its ARC design to compete head-on with other licensed microprocessors such as MIPS, SPARC, or ARM, primarily because of its limited feature set and tool support. Instead, the company is targeting ASIC developers who need to add a limited amount of programmability to an application-specific device such as a graphics accelerator or communications interface. This strategy also plays into ARC's strength: its extensibility.

Like any good RISC processor, ARC has an orthogonal set of 32 general-purpose 32-bit registers, as shown in Figure 1. The last three registers (r29–r31) hold link addresses for returning from branches or interrupts. The core's internally addressable resources include another 32 registers, of which only four (r60–r63) are defined. The remaining 28 are left for user extensions. The 24-bit loop-count register, r60, is used for loop overhead, similar to the CX register on x86 chips. A half-dozen registers for control and status are also defined.

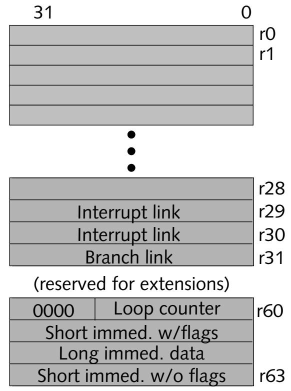

Figure 1. The ARC architecture includes 64 32-bit registers, of which the first 32 are mostly general-purpose in nature. An additional 4 registers are used for special cases, and 28 are currently undefined.

Instruction Set Trimmed to the Bone

ARC uses a fixed, 32-bit word to encode all instructions, of which the five most significant bits encode the operation, as Figure 2 shows. This definition allows a total of 32 instructions. Of these, 16 base instructions (with 10 variations) are permanently defined, and the remaining 16 are left to the user's imagination. The 26 architecturally defined instructions, listed in Table 1, use opcodes 0x00 through 0x0F.

Another five bits are used to encode conditions for the conditional instructions. This supports 32 combinations of conditions, of which the first 16 are defined, including the usual tests for zero, sign, overflow, and carry. The 16 user-defined condition codes could be used to test external signals or the status of additional function units.

Like ARM, ARC can conditionally execute each instruction and can also optionally set the condition-code flags after each instruction. A single bit (F) in the instruction word selects the latter option. For branches, two bits (NN) conditionally enable or disable the instruction in the branch-delay slot.

Most ARC instructions use three-operand addressing; with a 64-register set, 6 bits are required for each register address, usurping 18 bits from the instruction word. Instructions taking immediate data substitute the data literal for the conditional-execution and flag fields.

Bounding Registers Allow Zero-Overhead Loops

ARC takes an unusual approach to looping constructs. Rather than specifying the target address as part of the loop instruction, as conventional processors do, ARC instead currentlybounds the parameters of a loop through two special-purpose registers and r60. The two special registers are loaded with the start and end addresses of the loop, respectively, and r60 is loaded with the iteration count.

The LPcc instruction loads the special registers and initiates a loop if its condition is true; otherwise, execution branches to a specified offset. Execution repeats between the defined start and end addresses until the loop count is exhausted or until a conditional instruction exits the loop.

The lone conditional branch instruction takes a signed 20-bit offset, allowing forward or backward branches of up to one megabyte in either direction. Indirect branching (that is, specifying the target address using a register) is not

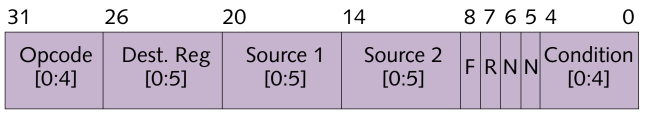

Figure 2. Most ARC instructions use three-operand addressing, with 9-bit signed immediate data taking the place of the condition, flag, reserved, and delay-slot bits.

supported, although indirect jumps are. Branch-and-link stores the flags and a 24-bit return address in the branch-link register, r31. Nesting of branches requires software-controlled stack management.

Loads and stores can use any general-purpose register plus a signed 8-bit displacement as an address pointer. Loads can also use a 32-bit displacement. Register scoreboarding allows nonblocking loads, while a read buffer handles up to four outstanding load operations before stalling the pipeline. Operand size (byte, word, double) can be specified; short operands can be sign-extended by the load instruction or with the popular SEX operation.

Technically, there is no MOV instruction—the MOV mnemonic doubles for the AND opcode, with the destination register ANDed onto itself. Likewise, the NOP is actually an XOR with all bits set; logical and arithmetic shifts to the left are handled by ADD instructions.

Four-Stage Pipeline Supplies Sufficient Speed

Argonaut defines the ARC core as a four-stage pipeline (fetch, decode, execute, write-back), with operands fetched and aligned during the second stage. Condition codes, as well as arithmetic and logical results, are written back during the final stage. Like many processors, the result write-back causes a dependence in the pipeline. Specifically, conditional flow-control instructions cannot follow any instruction that sets the condition codes.

Even with only about two-dozen defined opcodes, ARC has a fairly complete instruction set. Still missing are multiply, multiply-accumulate, and divide operations, logical comparison, multiple-bit shifts and rotates, and a rotate-left operation that doesn't use the carry bit. None of these operations is strictly necessary, of course, but would ease programmers' tasks and lead to denser object code.

Opcode	Mnemonic	Description
00	LD	Load indirect, register + register
01	LD	Load indirect, register + signed offset
01, 10	LR	Load from auxiliary register
02	ST	Store indirect, register + signed offset
02, 10	ST	Store to auxiliary register
03, 00	FLAG	Set condition-code flags
03, 01	ASR	Arithmetic shift right 1
03, 02	LSR	Logical shift right 1
03, 03	ROR	Rotate right 1
03, 04	RRC	Rotate right through carry 1
03, 05	SEX	Sign-extend 8 → 32
03, 06	SEX	Sign-extend 16 → 32
03, 07	EXT	Zero-extend 8 → 32
03, 08	EXT	Zero-extend 16 → 32
04	Bcc	Conditional branch
05	BLcc	Conditional branch and link
06	LPcc	Conditional enter loop mode
07	Jcc	Conditional jump
08	ADD	Add
09	ADC	Add with carry
0A	SUB	Subtract
0B	SBC	Subtract with borrow
0C	AND	Logical AND
0D	OR	Logical OR
0E	BIC	Logical AND-invert
0F	XOR	Logical exclusive-OR
10–1F	reserved	User-defined extensions

Table 1. The basic ARC instruction set uses just 16 major opcodes; opcode extensions, where used, replace the Source 2 field. Two addressing modes are used for loads and one for stores. Sixteen opcodes are reserved for enhancements.

For More Information

Argonaut Technologies (London, England) can be reached by telephone at +44.181.358.2970, by fax at +44.181.358.2920, by e-mail at *techsales@argonaut.com*, or on the Web at *www.argonaut.com.*

Third-Party Support Still Lacking

As a company, Argonaut Technologies has some history behind it. The company is a wholly owned subsidiary of Argonaut Software, which has developed graphics software and hardware for a number of companies, including Nintendo. The company has also carried out some preliminary work with Chip Express (San Jose, Calif.) and LSI Logic. For software support, the company has signed GNU tool-chain vendor Cygnus.

The extensible register file and instruction set make tool support problematic. Compilers, for example, must necessarily support only the basic ARC definition with 16 basic opcodes and 32 general-purpose registers. Any additions to that feature set would, by definition, be application-specific and nonportable.

ARC's biggest advantage to the designer is the close-in access to the core. Because it is distributed as a VHDL model rather than a physical macro, new registers and special function units can be grafted on, taking advantage of decoding logic and data paths already present. Such core-level modifications are not possible with ARM, SPARC, or MIPS designs and allow very close cooperation between the CPU and user logic. For example, an ARC core can share data with a new function unit without explicitly loading or storing operands over an external bus. Data can be captured from the internal data path and status monitored with new condition codes. In a conventional core-based ASIC design, the same operation would use traditional load and store instructions, which are much slower. With ARC, data transfers within the core happen almost for free.

Because Argonaut has no schemes to become a mainstream CPU core supplier, the limitations of scarce development-tool support may not affect ARC's acceptance. For ASIC designers who simply want to contract out the time, risk, and effort of designing a basic programmable core, ARC presents a viable alternative to some of the better-known embedded CPU designs. But with little third-party support, no discernible advantage in die size, and unknown performance, Argonaut will have a tough time making ARC into anything other than a niche curiosity. M