MICROPROCESSOR REPORT

THE INSIDERS' GUIDE TO MICROPROCESSOR HARDWARE

VOLUME 9 NUMBER 4 MARCH 27, 1995

Thumb Squeezes ARM Code Size

New Core Module Provides Optimized Second Instruction Set

by James L. Turley

Seeking to shoulder the trophy for best code density, Advanced RISC Machines (ARM) has taken a unique approach, adding an entirely new instruction set to its ARM7 processor core. The new feature, inevitably called Thumb, seeks to shoehorn the 36 most commonly used instructions into a 16-bit instruction format. A Thumb-equipped processor can execute both 32-bit ARM code and 16-bit Thumb code, though the two instruction sets cannot be directly intermixed.

Thumb addresses the issue of code density in an innovative way. In embedded systems, the cost of ROM and RAM can make up a significant portion of the cost of the system, so reducing the size of compiled object code, or improving code density, is an important consideration. ARM's engineers evaluated different approaches, including actually compressing finished programs with a Huffman or PKZIP-type algorithm and then adding real-time decompression hardware. The ultimate solution is much simpler: ARM defined a second instruction set.

Thumb is not a new processor but an optional macrocell for ARM7-based designs. The macrocell adds an instruction predecoder (which the company calls an "decompressor") and associated control logic. The new instruction set is not complete: exception handling and MMU configuration can't be performed with Thumb code, relying instead on the original 32-bit instruction set. Thumb also limits program access to eight registers.

The results are encouraging. Object code averages 25–35% smaller than comparable ARM code. Thumb code also performs better than ARM over a 16-bit bus.

The announcement comes on the heels of the company's recent agreement with Digital Semiconductor to develop a new generation of high-end chips *(see **0902MSB.PDF**)*, but the two developments are not related. While the StrongArm venture is aimed at high-end designs, Thumb is intended to elbow into the lower end of the market, where 16-bit microprocessors traditionally have an advantage in code density.

New Design Thumbs Nose at 16-Bit RISC

In selecting which 36 lucky instructions would be part of the new instruction set, ARM's designers considered three criteria: those instructions that do not need 32 bits (i.e., do not substantially degrade through 16-bit encoding); those that are used most often; and those that are required by compilers to implement reasonable object code. (Exclusive-OR instructions, for example, do not appear frequently but are invaluable when needed.)

Using an iterative process to arrive at the final instruction set, code samples were recompiled and the resulting object sizes compared. By removing some instructions and including others, the company arrived at an acceptable performance/code-size tradeoff.

ARM's engineers could afford to be picky about which instructions are implemented by Thumb. Major functions are ignored by relegating them to the standard 32-bit ARM instruction set. System-control functions, for example, are not supported, nor are coprocessors, exception handling, or the upper half of the register set. Thumb, therefore, is not a complete instruction set in the true sense but rather a collection of frequently used instructions in shorthand form. The entire Thumb instruction set appears in Table 1.

Tiny Instructions Sacrifice Generality

Many sacrifices had to be made to fit the ARM instruction set—which is already among the densest of any 32-bit RISC architecture—into half the size. The first amputation was ARM's most unusual feature: its conditional execution of every instruction (see MPR 12/18/91, p. 11). ARM chips also have the ability to shift or rotate one operand as an intrinsic part of any instruction; this feature, too, was dropped for space reasons. Finally, Thumb instructions always update the processor's status flags, a previously optional step and another move toward a more conventional programming model.

The removal of the conditional-execution feature is the most profound change in the architecture. Normally,

ARM does not use conditional branch instructions; every instruction is conditional, virtually eliminating branches around small code fragments. Thumb, on the other hand, has a traditional branch on condition (Bcc) instruction familiar to programmers of other microprocessors.

Logical shift and rotate operations come for free in the original ARM instruction set. Any three-operand instruction (AND, EOR, etc.) can shift or rotate one source operand before it is used. By simply specifying a null logic operation or a move in place, shifts or rotates can be executed alone. The additional bits required to encode these options were not available to the Thumb designers, so discrete arithmetic (signed) and logical (unsigned) shift and rotate instructions were added.

There are no changes to the ARM7's 32-bit core, so $32 \times 32 \rightarrow 32$-bit multiply operations are untouched, but the multiply-accumulate instruction is not supported in Thumb. Neither are atomic memory transactions, coprocessor operations, or the two reverse-subtract (RSB, RSC) instructions. A new NEG instruction performs the same function as a reverse-subtract from zero.

Mnemonic	Description	ARM Equivalent
ADD	Add	ADDS Rd, Rs, Rn
ADC	Add with Carry	ADCS Rd, Rd, Rs
SUB	Subtract	SUBS Rd, Rs, Rn
SBC	Subtract with Carry	SBCS Rd, Rd, Rs
MUL	Multiply	MUL Rd, Rs, Rd
AND	Logical AND	AND Rd, Rs, Rn
ORR	Logical OR	ORRS Rd, Rd, Rs
EOR	Exclusive-OR	EORS Rd, Rd, Rs
LSL	Logical Shift Left	MOVS Rd, Rs, LSL Rs
LSR	Logical Shift Right	MOV Rd, Rd, LSR Rs
ASR	Arithmetic Shift Right	MOVS Rd, Rd, ASR Rs
ROR	Rotate Right	MOVS Rd, Rd, ROR Rs
CMP	Compare	CMP Rd, #offset8
NEG	Negate	RSBS Rd, Rs, #0
CMN	Compare Negative	CMN Rd, Rs
BIC	Bit Clear	BICS Rd, Rd, Rs
TST	Test Bits	TST Rd, Rs
B	Branch Unconditional	B {label}
Bcc	Branch Conditional	B[xx] {label}
BL	Branch and Link	BL {label}
BX	Branch and Exchange	n/a
MOV	Move Register/Register	MOVS Rd, #offset8
MVN	Move and Invert	MVNS Rd, Rs
LDR	Load Word	LDR Rd, [PC, #imm5]
LDRB	Load Byte	LDRB Rd, [Rb, #imm5]
LDRSB*	Load Signed Byte	n/a
LDRH*	Load Halfword	n/a
LDRSH*	Load Signed Halfword	n/a
LDMIA	Load Multiple	LDMIA Rb!, {Reg List}
STR	Store Word	STR Rd, [Rb, Ro]
STRB	Store Byte	STRB Rd, [Rb, Ro]
STRH*	Store Halfword	n/a
STMIA	Store Multiple	STMIA Rb!, {Reg List}
PUSH	Push Registers	STMDB R13!, {Reg List}
POP	Pop Registers	LDMIA R13!, {Reg List}
SWI	Software Interrupt	SWI value8

Table 1. A complete list of Thumb instructions shows that, while the instruction set is very small, most common functions are supported. *New instructions with ARM Version 4.

The load-multiple and store-multiple instructions are still available but are hard-coded to use the post-increment addressing mode. New PUSH and POP instructions make up for the lack of stack addressing modes, standing in for the load/store-multiple instructions with pre-decrement and post-increment addressing, respectively.

Hardware Translation Adds No Delays

The Thumb module works by reconstituting incoming Thumb opcodes into their matching ARM instructions. It does this through a simple hardwired lookup table. Because there are so few Thumb opcodes, and because they all map one-to-one to ARM instructions, the lookup PLA is fairly small. Thumb adds approximately 3,000 transistors to an ARM7 core, representing about 5% of the total core area, or less than 1 mm^2 in a 0.8-micron three-layer-metal process.

The Thumb-to-ARM conversion happens on the fly as part of the second pipeline stage. At current clock speeds, enough time is available in the decode stage to add the extra logic without impacting pipeline throughput or latency.

After the opcode bits are converted, register references are extracted from the Thumb instruction and placed in their proper positions in the ARM instruction, as shown in Figure 1. Most Thumb instructions accept two operands, while ARM instructions generally take three; the missing source register is created by duplicating the destination operand, creating a destructive two-operand operation—a classic CISC feature. If an immediate value is present, it is zero-extended to fill the larger instruction field. When the fully reconstituted 32-bit instruction makes its way through the pipeline, the standard ARM instruction decoder is none the wiser.

Some operations are encoded in as few as 4 opcode bits; others take as many as 8. Instructions like ADD that are used frequently and have many formats use a variable encoding scheme. The add-immediate instruction uses 5 opcode bits, leaving 3 bits for a destination register and 8 bits for the immediate value. A three-register ADD, on the other hand, uses a 7-bit opcode to distinguish it from the add-immediate version and sacrifices the remainder of the operand field to identify two source registers. The SUB (subtract) instruction works the same way. Add and subtract are the only Thumb instructions that retain support for three-operand operations.

A new CPU status bit controls a multiplexer, routing incoming instructions through the Thumb predecoder or directly into the usual ARM decode logic. Changing the bit requires a special branch-and-exchange (BX)

instruction, so ARM code and Thumb code must be segregated into separate procedures. Interrupts and other exceptions switch the processor to 32-bit mode, allowing full use of the processor's resources. ARM and Thumb routines can call each other freely without confusing the processor.

Effects on Performance Are Complex

The presence of a Thumb preprocessor has a neutral effect on the performance of cached code. Because Thumb instructions are cached while still in their compressed form, twice as many Thumb instructions as ARM instructions fit in the cache. Although more instructions are needed, they are only half as long, so the overall size of most programs is reduced. This results in a larger proportion of program code in the cache.

This benefit is neutralized, however, by the greater number of instructions required and by the addition of conditional branches, which cause extra pipeline breaks. The net performance is equal, to within about 5%, between ARM code and Thumb code when both are cached.

The effect of memory latency on Thumb performance is particularly intriguing. Given an ideal system with a 32-bit memory bus and no wait states, and ignoring the effects of cache, Thumb code runs about 20% slower due to the increased number of instructions required to perform routine tasks. However, after adding just one wait state to external bus cycles, ARM and Thumb run neck and neck. Adding more wait states gives Thumb an increasing edge.

Thumb has a leg up after only a few wait states because the processor can fetch two instructions during every 32-bit bus access. Although the latency affects both ARM and Thumb equally for the first instruction, a processor fetching Thumb code effectively gets every second instruction for free. The minimum ARM7 bus transaction lasts only one clock cycle, so one wait state equates to a half-speed bus—the point at which ARM and Thumb cores perform almost equally. Additional latency penalizes ARM code more severely.

The same advantage accrues to other microprocessors that have buses wider than their instruction size. The R4650, for example, has a 64-bit bus, allowing it to perform the same trick as Thumb but on a double-wide scale. Hitachi's SH7604 and new SH7708 *(see **090302.PDF**)* are similar to Thumb in that they fetch 16-bit code over a 32-bit bus.

The effect is even more pronounced in lower-cost configurations. In systems with a 16-bit bus, Thumb processors can fetch an instruction in a single cycle. As Figure 2 shows, Thumb code actually runs faster than equivalent ARM code over an 8-bit or 16-bit bus.

Halving the instruction size doesn't double the number of instructions. For routines that can be executed entirely in Thumb code, the space savings are significant.

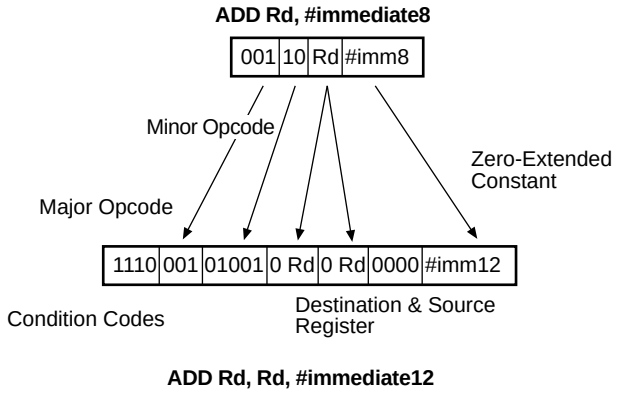

Figure 1. The Thumb instruction decompression logic expands opcodes and register references into their 32-bit ARM equivalents.

In reality, programs may have to mix ARM and Thumb code. Table 2 shows that, for comparatively real-world applications from the SPECint suite, substantial overall space savings can be found.

When Does It Pay Off?

Not all ARM instructions can be represented in Thumb's shorthand notation, forcing some routines to be coded in the 32-bit instruction set. Any routine that requires a 32-bit ARM instruction has to be written entirely in ARM code, or branch to ARM code. With no rule of thumb, it is difficult to estimate the point at which it becomes effective to recompile existing ARM programs.

Frequently used loops might benefit from recompilation, branching to and from a 32-bit handler. Several factors will affect the final result. First, the BX instruction itself has a latency of three clock cycles, which would not be incurred if the code were simply in line. Two BX instructions would typically be needed, one at each end.

Second, Thumb code cannot take advantage of most of the quirks that make the ARM instruction set attractive in the first place, namely its conditional execution and zero-latency shifts and rotates. Thumb code must execute discrete shift or rotate instructions, perform ex-

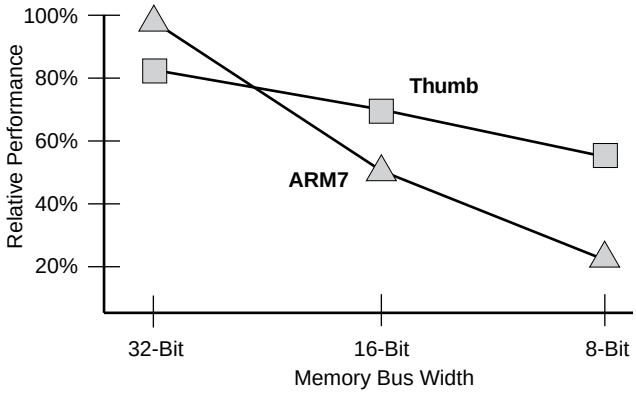

Figure 2. Comparing ARM7 cores with and without Thumb shows the effects of memory bus width on performance. The 16-bit Thumb instruction set surpasses ARM in low-cost systems. (Source: ARM).

Price & Availability

The first Thumb-compatible processor core, the ARM7TDMI, will be available in June. For more information, contact Advanced RISC Machines (Cambridge, U.K.) at 44.1223.400.400; fax 44.1223.400.410, or ARM North America (Los Gatos, Calif.) at 408.399.5199; fax 408.399.8854.

plicit comparisons, and branch conditionally based on the outcome. Most routines will require more instructions, even if they use fewer bytes of storage.

Third, the Thumb-encoded routine may or may not execute at the same speed as a similar algorithm in 32-bit mode. As shown previously, if the code has a low cache-hit rate, and memory latency is more than three cycles, the Thumb code may actually execute more quickly, despite the added steps. For systems that need all the speed they can get, the best performance still comes from 32-bit ARM code running over a no-wait-state, 32-bit bus.

New Version Handles Four New Instructions

ARM's nomenclature assigns a letter suffix to each optional core module. An ARM7 with the fast multiplier becomes ARM7M, debug support adds a D, and in-circuit emulator support is identified by the letter I. In that taxonomy, Thumb will be tagged with a T suffix. The first Thumb-compatible core will be an ARM7TDMI—a fully equipped processor core.

What's not so obvious is that the ARM7TDMI core will also be the first to implement the ARM Version 4 instruction set. This new definition adds four instructions and a privileged system mode. System mode is intended for operating-system tasks, allowing the processor to execute privileged instructions while sharing the user-mode register set.

The new instructions add support for signed byte and halfword (16-bit) operands, something ARM previously lacked. The LDRSB instruction sign-extends a byte from memory into a general-purpose register. Likewise, LDRSH sign-extends halfwords, while the new LDRH in-

Test	Thumb	ARM7	386	8088	68020	SPARC
eqntott	10,608	16,768	17,640	19,106	20,542	22,256
Ratio	0.63	1.00	1.05	1.14	1.23	1.33
xlisp	26,388	40,768	28,097	29,401	46,746	44,648
Ratio	0.65	1.00	0.69	0.72	1.15	1.10
espresso	72,596	109,923	125,686	137,194	131,854	142,752
Ratio	0.66	1.00	1.14	1.25	1.20	1.30

Table 2. A comparison of object code size for three SPECint benchmarks shows that even with an increased number of instructions, Thumb's code density improves significantly over ARM7. (Source: ARM, Micrologic Solutions)

Sharp Debuts ARM Controller

Sharp Electronics is using an ARM7 core as the basis for its new integrated processor for handheld devices. The chip includes 4K of cache, 2K of data SRAM, an LCD controller, a DRAM controller, and serial and parallel interfaces in a 160-lead TQFP package. Sharp is hoping to attract makers of handheld instruments, portable medical equipment, and personal communicators with its new device.

While Sharp's LH77790 chip does not integrate the new Thumb module, it does have a 16-bit external data bus. The on-chip 4K unified cache should keep the part running happily at its 25-MHz peak, though off-chip accesses will slow its execution rate considerably. Apart from the 2K of RAM, there is no on-chip memory, so applications code will have to be fetched—at least initially—from off-chip ROM.

The LH77790 includes a memory controller in lieu of a conventional ARM-style system bus. The controller drives five programmable chip selects and includes simple SRAM/EPROM control signals for WE and OE. Support for page-mode DRAMs is also included. Harking back to the now-forgotten ARM2 and ARM3 chips, the external address bus is only 26 bits wide, enough for 64M of external memory. For a handheld device, this restriction should not be too grievous. The 16-bit data bus reduces cost but badly affects the performance of programs that miss the cache.

The most complex feature of the chip is its monochrome LCD controller. Suitable for so-called half-VGA LCD displays, it supports programmable resolution up to 480 × 320 with four gray shades (2-bit pixel depth). Sharp, by no coincidence, manufactures several such LCD panels. Five programmable pulse-width modulation outputs are also included, two of which could be used for software control of the LCD brightness and contrast.

Three 16-bit counter/timers, a 24-bit parallel I/O port, and three 16C450-compatible serial ports round out the I/O section. One of the serial ports supports Irda infrared communications.

Although the chip has a 32-bit ARM7 core and a controller for a reasonably sized LCD screen, its narrow external bus and small cache make it a little underpowered for upcoming PDA applications. Sharp has correctly positioned it for instrumentation, pocket organizers, and point-of-sale terminals instead.

This is the second ARM-based device to come from Sharp. The company entered the market in early 1994 with its LH74610, an ARM610 chip that appears in Apple's Newton MessagePad 100 and 110 PDAs *(see **0803MSB.PDF**)*.

Samples of the LH77790 are expected in July. Pricing has not yet been disclosed. For more information, contact Mike Roberts or Len Elias at Sharp Electronics Technology, (Camas, Wash.) phone 360.834.8791 or 360.834.8966; fax 360.834.8611.

struction zero-extends halfwords. ARM chips could always store individual bytes; now the STRH instruction permits storing halfwords without first extending them to 32 bits. All four new instructions support the usual assortment of addressing modes.

The halfword loads and store make the new ARM core more efficient with 16-bit buses, an area that ARM is clearly targeting. Most signal-processing applications typically deal with 10–12-bit words, making 16-bit loads and stores valuable. Although the new Version 4 instructions are not dependent on a Thumb decoder, both are aimed at attracting the same kind of designers.

ARM Flexes Design Muscle

The Thumb module is a unique approach to improving code density. Content with neither a 32-bit instruction set nor a 16-bit one, ARM chose to offer its customers both. As mentioned previously, Thumb is not really a full-featured instruction set, so the company was able to skimp on the system-level instructions and instead implement only the most popular functions for the tightest code.

It seems likely that most, if not all, future ARM cores will come with the Thumb predecoder on them as a matter of course. It costs very little in silicon, and the benefits can be significant. Including the module also avoids a potential quagmire of compatibility issues between ARM processors that support Thumb and those that don't. With or without Thumb, ARM processors can still execute 32-bit ARM code.

Compared with other microprocessors that have 16-bit instructions, Thumb sticks out. Hitachi's SH series is notable for toeing the line on 16-bit instructions and has enviable code density to show for it. Conversely, because the SH doesn't have a "real" 32-bit instruction set to fall back on, it must squeeze everything into 16 bits. Losing many of those precious opcode bits to system-control functions forces compromises, like the SH's lack of variable shift or rotate instructions and its implicit register usage. Consequently, the SH can be an awkward chip for assembly-language programmers.

NEC's V851 nominally has 16-bit instructions but makes frequent use of 16-bit extension words as well. This is also true of the entire 680x0 family. As Motorola and NEC have found, the ideal instruction length seems to vary somewhere around 24 bits, and leavening instructions with extenders was the only practical way to achieve that size.

Plenty of chips have privileged or special modes that use different instructions, but no other microprocessor today has such a marked difference between its default and its special-purpose instruction set. Such special instructions usually serve control-and-configuration purposes; they're necessary overhead and provide little advantage to the programmer. Thumb, on the other hand, exists only to make code denser.

One obvious use for Thumb is in the upcoming generation of PDAs from Apple and others. Newton carries several megabytes of code in expensive on-chip ROM, and compacting that code would make the product more cost-effective. Reducing the bus width to 16 bits is also appealing for lower-end devices. Cirrus Logic, VLSI, Sharp, and other ARM licensees with a history of supplying core logic chips to Apple should be among the first to capitalize on the benefits of Thumb and the Version 4 extensions (see sidebar).

Two interesting trends are evident here. First, while operating in Thumb mode, the processor is not really executing its own native instruction set. Like the P6, K5, and Nx586, ARM is translating binary code on the fly into a form it can execute. Second, Thumb is starting to show the early signs of adopting CISC principles. Like many near-RISC chips before it, Thumb uses destructive two-operand instructions, special-purpose addressing modes that are not orthogonal, and a small register set.

One of the most compelling benefits of Thumb is that programmers are not required to use it, allowing them to tinker with recompiling their existing applications, looking for the best MIPS/byte tradeoff.

Many microprocessors try to find the best balance among performance, capability, and code size. By their nature, embedded processors necessarily have to make compromises. By not standing on convictions about what a RISC processor should be, ARM has pointed out a unique approach by offering two architectures in one. ♦

