ll: Exploring the Limits of Code Density

Vincent M. Weaver
University of Maine
vincent.weaver@maine.edu

May 26, 2017

Abstract

This document is a continuation of my code density work as described in our ICCD'09 [2] paper. This is just a summary document meant to accompany the project sourcecode. Included are updated versions of the graphs and tables from the original code density work, updated as new architectures are added.

I hand-assemble a simple benchmark on a number of architectures with the end goal being the smallest code size. A comparison can then be made of the code density of the architecture. The benchmark is small and simple, so may not give a full accounting of code density for larger benchmarks, but picking a larger benchmark would make the hand-coded assembly task much larger. The benchmark does have some useful routines in it, such as LZSS compression [3, 1], string concatenation, and integer to string conversions.

1 Additional Findings since ICCD'09

In theory the new x86 SSE4 string instructions should be great for doing some of these operations, such as strlen or strcat. I could not find a way to use them to find the results in fewer bytes than the discrete instructions.

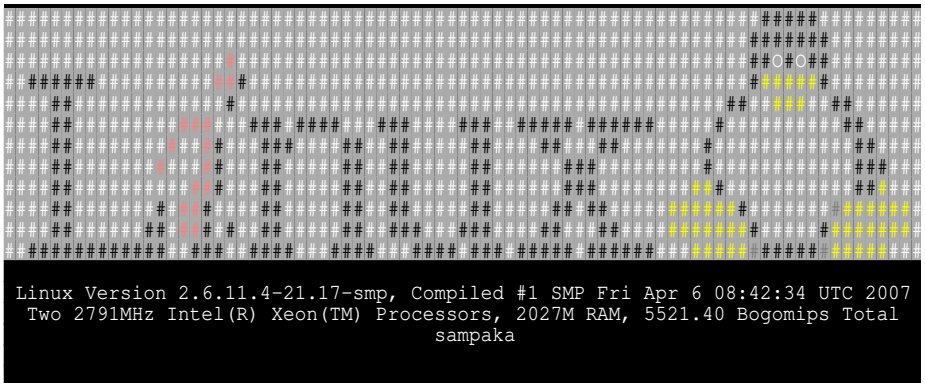

Figure 1: Sample output from the `linux_logo` benchmark

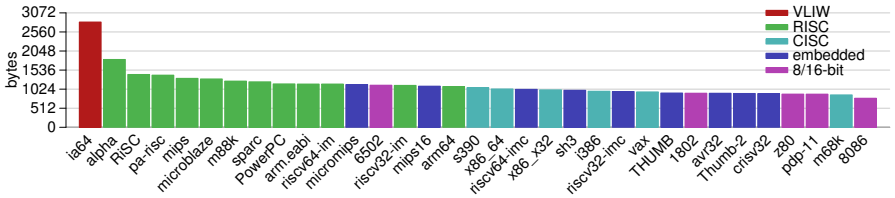

Figure 2: Total size of benchmarks (includes some platform-specific code, so does not strictly reflect code density)

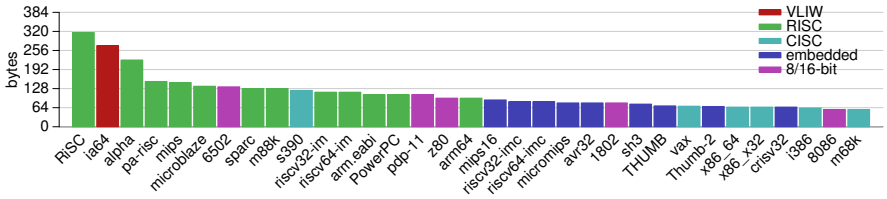

Figure 3: Size of LZSS decompression code

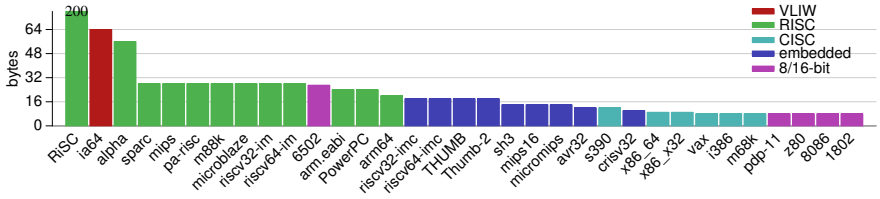

Figure 4: Size of string concatenation code (machines with auto-increment addressing modes and dedicated string instructions perform better). 6502 behaves poorly as it lacks an increment-16-bit-register instruction.

Table 1: Summary of investigated architectures

Type	arch	endian*	bits	instr len (bytes)	op args	GP int regs	unaligned ld/st	auto-inc address	hw div	stat flags	branch delay	predi-cation
VLIW	IA64	little	64	16/3†	3	127,zero	no	yes	no	yes	no	yes
RISC	Alpha	little	64	4	3	31, zero	no	no	no	no	no	no
	ARM	little	32	4	3	15,PC	no	yes	no	yes	no	yes
	ARM64	little	64	4	3	31,zero	yes	yes	yes	yes	no	no
	m88k	big	32	4	3	31,zero	no	no	Q only	no	optional	no
	MicroBlaze	big	32	4	3	31,zero	no	no	Q only**	no	optional	no
	MIPS	big	32/64	4	3	31,hi/lo,zero	yes**	no	yes	no	yes	no
	PA-RISC	big	32/64	4	3	31,zero	no	no	part	no	yes	no
	PPC	big	32/64	4	3	32	yes	yes	Q only	yes	no	no
	riscv-im	little	64	4	3	31	yes	no	yes	no	no	no
	RiSC	big	16	2	3	7,zero	no	no	no	no	no	no
	SPARC	big	32/64	4	3	63-527,zero‡	no	no	Q only	yes	yes	no
CISC	m68k	big	32	2-22	2	16	yes	yes	yes	yes	no	no
	s390	big	32/64	2-6	2	16	yes	no	yes	yes	no	no
	VAX	little	32	1-54	3	16	yes	yes	yes	yes	no	no
	x86	little	32	1-15	2	8	yes	yes	yes	yes	no	no
	x86_64	little	32/64	1-15	2	16	yes	yes	yes	yes	no	no
Embedded	AVR32	big	32	2	2	15,PC	yes	yes	yes	yes	no	no
	CRISv32	little	32	2-6	2	16,zero,special	yes	yes	part	yes	yes	no
	riscv-imc	little	64	2-4	2-3	8/31	yes	no	yes	no	no	no
	mips16	big	32/64	2-4	2/3	8/31	yes	no	yes	reg	yes	no
	micromips	big	32/64	2-4	2/3	8/31	yes	no	yes	no	yes	no
	SH3	little	32	2	2	16,MAC	no	yes	part	yes	yes	no
	THUMB	little	32	2	2	8/15,PC	no	no	no	yes	no	no
	THUMB-2	little	32	2-4	2-3	8/15,PC	no	yes	no	yes	no	yes
16-bit	8086	little	16	1-15	2	8	yes	yes	yes	yes	no	no
	PDP-11	little	16	2-6	2	6,sp,pc	no	yes	yes**	yes	no	no
8-bit	1802	big	8	1-3	1	17	n/a	yes	no	yes	no	yes
	6502	little	8	1-3	1	3	yes	no	no	yes	no	no
	z80	little	8	1-4	2	18	no	lim	no	yes	no	no

* on the machine we used † 16-byte bundle has 3 instructions ‡ register windows, only 32 visible ** many implementations

Table 2: Correlations of architectural features to overall binary size

Correlation Coefficient	**Architectural Parameter**
0.9061	Minimum possible instruction length
0.8315	Number of integer registers
0.6975	Virtual address of first instruction
0.5834	Architecture has a zero register
0.4433	Bit-width
0.4354	Number of operands in each instruction
0.3181	Predicated instructions
0.2120	Year the architecture was introduced
0.0448	Branch delay slot
0.0316	Machine is big-endian
-0.0331	Maximum possible instruction size
-0.2561	Auto-incrementing addressing scheme
-0.2992	Hardware status flags (zero/overflow/etc.)
-0.3336	Hardware divide in ALU
-0.3684	Unaligned load/store available

Table 3: Correlations of architectural features to lzss size

Correlation Coefficient	**Architectural Parameter**
0.6428	Architecture has a zero register
0.5960	Minimum possible instruction length
0.5017	Number of integer registers
0.4291	Number of operands in each instruction
0.2707	Virtual address of first instruction
0.1929	Machine is big-endian
0.1423	Bit-width
0.1295	Predicated instructions
0.0813	Year the architecture was introduced
-0.0151	Branch delay slot
-0.2023	Maximum possible instruction size
-0.4287	Hardware divide in ALU
-0.4526	Auto-incrementing addressing scheme
-0.4680	Unaligned load/store available
-0.5062	Hardware status flags (zero/overflow/etc.)

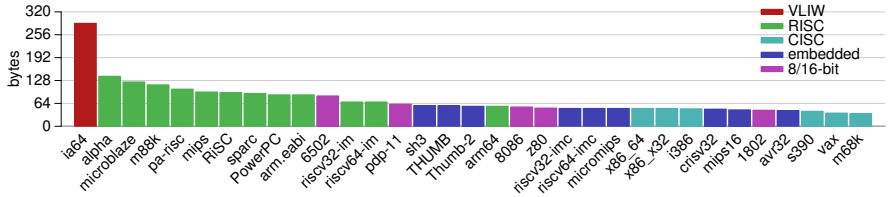

Figure 5: Size of string searching code (unaligned load instructions help, since four bytes at arbitrary offsets can be compared at once. CISC architectures as well as arm64, avr32 and MIPS benefit)

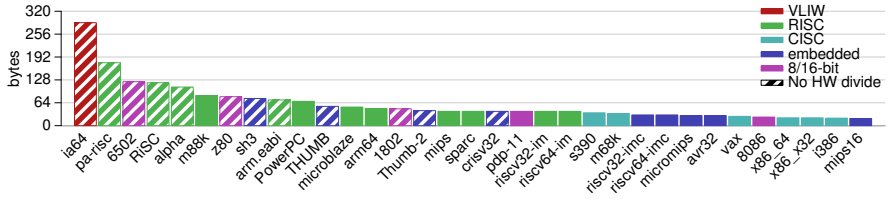

Figure 6: Size of integer printing code (hardware divide helps code density)

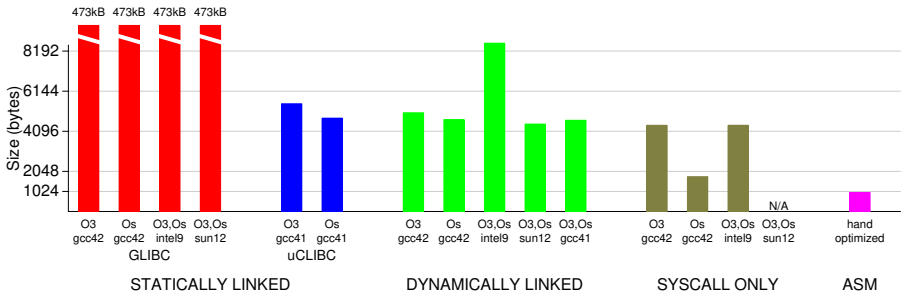

Figure 7: Size comparison

References

[1] J. Storer and T. Szymanski. Data compression via textual substitution. *Journal of the ACM*, 29:928–951, 1982.

[2] V.M. Weaver and S.A. McKee. Code density concerns for new architectures. In *Proc. IEEE International Conference on Computer Design*, pages 459–464, October 2009.

[3] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.