tinyRV：一种完备的RISC-V定制压缩指令集

兰州大学异步电路与系统实验室

陈名书

2024年10月12日

目录

1. 研究背景

对大多数处理器核心而言，发射指令流和Cache失效是能量耗散的两个重要原因[1]。例如，在DEC StrongARM-110中，指令地址转换和缓存访问占芯片功耗的36%[2]。在另一项研究中，指令Cache的访问本身就消耗了五级流水处理器40%的能量[3]。并且对于面向控制的嵌入式处理器，最终的电路的很大一部分用于指令Cache，如[2]中Icache的面积占整个处理器25%的面积，而[3]中则占约37.5%。而压缩指令集能够提高代码密度，一方面能够**降低指令缓存的面积**，一方面减低指令Cache失效率，**减少**由指令发射和指令Cache失效而引起的**能耗损失和性能降低**。特别在一些芯片面积要求严苛或者指令立即数范围较小时具有很大的优势。

TABLE III
SIMULATED POWER DISSIPATION BY SECTION

ICACHE	27%
IBOX	18%
DCACHE	16%
CLOCK	10%
IMMU	9%
EBOX	8%
DMMU	8%
Write Buffer	2%
Bus Interface Unit	2%
PLL	< 1%

[2]中各模块功耗

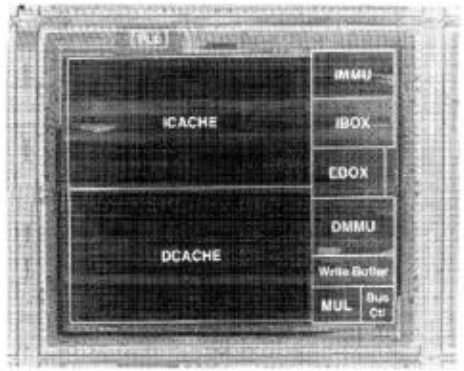
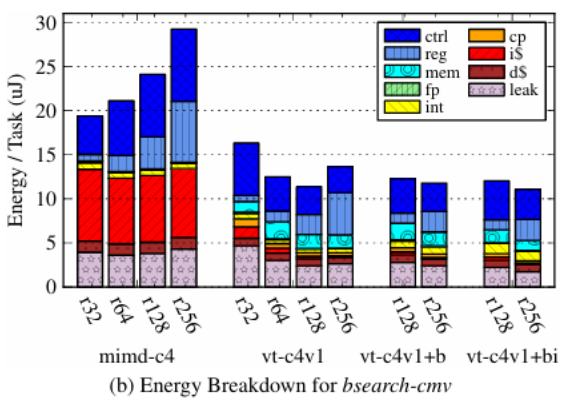


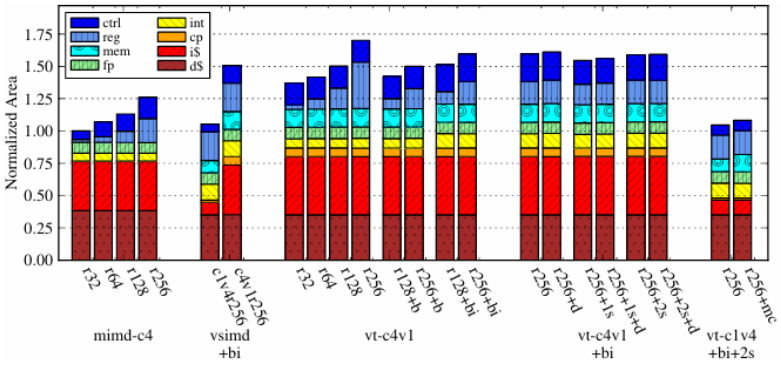
Fig. 1. Chip photo with overlay.

[2]的芯片版图



(b) Energy Breakdown for *bsearch-cmv*

[3]中各模块功耗



(a) Area Breakdown for Evaluated Tile Configurations

[3]中各模块面积

[1] L. Villa, M. Zhang and K. Asanovic, "Dynamic zero compression for cache energy reduction," Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000, Monterey, CA, USA, 2000, pp. 214-220.
[2] J. Montanaro, R. Witek and K. Anne,. A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor. Digital Technical Journal, 9:49–62, January 1997.
[3] Y. Lee, R. Avizienis and A. Bishara. Exploringthe Tradeoffs between Programmability and Efficiency in Data-Parallel Accelerators. In Proceedings of the 38th Annual International Symposium on Computer Achitecture, ISCA '11. ACM, 2011.

1. 研究背景

RV32C实际上RISC-V指令集中的**高频**指令的简记，在指令译码之后按照非压缩指令来执行。因此，**单独的 RV32C 指令集无法完成所有的程序工作（不完备）**。并且，对于不需要使用浮点操作的小型系统，RV32C中的**浮点操作区间并没有使用**。

RV32E指令	操作
sll	x[rd]=x[rs1]<<x[rs2]
srl	x[rd]=x[rs1] >>u x[rs2]
sra	x[rd]=x[rs1] >>s x[rs2]
slt	x[rd]=x[rs1] <s x[rs2]
sltu	x[rd]=x[rs1] <u x[rs2]
beq	if (rs1 == rs2) PC+=sext(offset)
bne	if (rs1 != rs2) PC+=sext(offset)
blt	if (rs1 <s rs2) PC+=sext(offset)
bge	if (rs1 >=s rs2) PC+=sext(offset)
bltu	if (rs1 <u rs2) PC+=sext(offset)
bgeu	if (rs1 >=u rs2) PC+=sext(offset)
slti	x[rd]=(x[rs1] <s sext(imm)])
sltiu	x[rd]=(x[rs1] <u sext(imm)])
lb	x[rd]=sext(M[x[rs1]]+sext(offset)][7:0])
lh	x[rd]=sext(M[x[rs1]]+sext(offset)][15:0])
sb	M[x[rs1]+sext(offset)]=x[rs2][7:0]
sh	M[x[rs1]+sext(offset)]=x[rs2][15:0]
ecall	
mret	
csrrw	t=CSRs[csr]; CSRs[csr]=x[rs1]; x[rd]=t
csrrs	t=CSRs[csr]; CSRs[csr]=t | x[rs1]; x[rd]=t
csrrc	t=CSRs[csr]; CSRs[csr]=t & ~x[rs1]; x[rd]=t
csrrwi	t=CSRs[csr]; CSRs[csr]=zimm; x[rd]=t
csrrsi	t=CSRs[csr]; CSRs[csr]=t | zimm; x[rd]=t
csrrci	t=CSRs[csr]; CSRs[csr]=t & ~zimm; x[rd]=t

[15:13] / [1:0]	000	001	010	011	100	101	110	111
00	ADDI4SPN	FLD	LW	FLW	Reserve	FSD	SW	FSW
01	ADDI	JAL	LI	LUI/ ADDI16SP	MISC-ALU	J	BEQZ	BNEZ
10	SLLI	FLDSP	LWSP	FLWSP	J[AL]R/MV/ ADD/EBREA	FSDSP	SWSP	FSWSP
11								

为什么可以进行压缩指令的定制？

①E型指令集为整数指令集，其不包含浮点操作，因此C型指令集中的**浮点区间可以重新定制**。

②E型只需要16（4bit）个寄存器，因此C型指令中寄存器地址编号有多余空间**扩展功能码**。

③Reserve的区间可以使用，最低两位为**11的区间**可以使用。

1. 研究背景

基准程序集分析

- **miBench：** 免费的、具有商用代表性的嵌入式基准程序集，包含工业控制、消费设备、办公自动化、网络、安全以及电信等六大类别的测试集。
- **RVBench：** RISC-V官方提供的的开源基准测试程序集，包含dhrystone、快排、乘法以及内存拷贝等经典测试程序。

测试集类别	名称	二进制文件大小（KB）	指令条数（K）
miBench[1]	crc_32	8.1	3.06
	sha	8.6	3.25
	bitcnt	10.3	3.86
	pbmsrch_small	6.4	2.42
RVBench[2]	dhrystone	6.1	2.23
	memcpy	4.7	1.75
	multiply	4.8	1.79
	qsort	5.1	1.89

- **选择依据：**
  1. 整型运算
  2. 程序代码量较小
  3. 涵盖范围较广

- **gcc编译选项：**

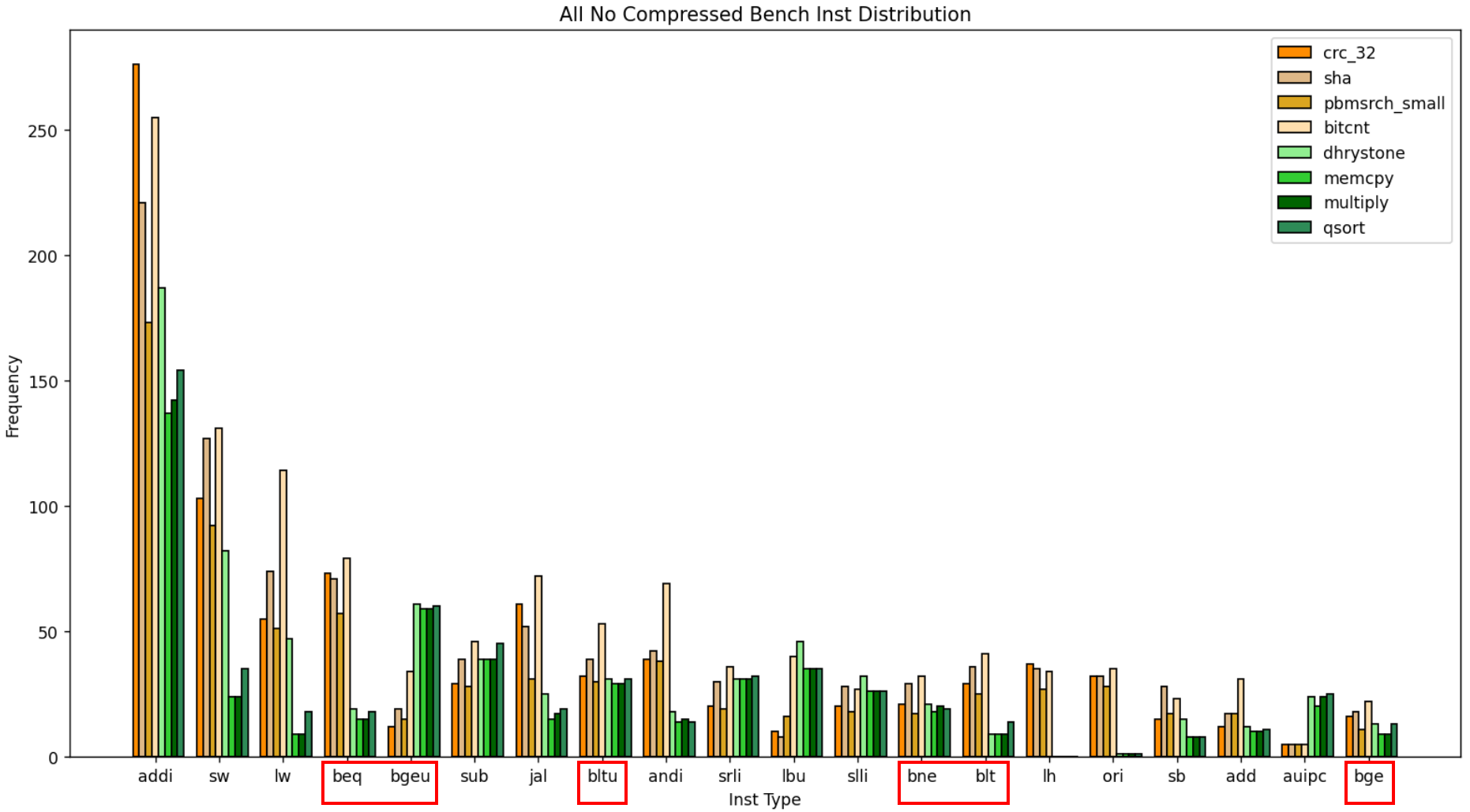
```
GCC := riscv32-unknown-elf-gcc
RISCV_ARCH := rv32ec_zicsr
RISCV_ABI := ilp32e
CFLAGS += -O2
LDFLAGS += -Wl,--gc-sections
```

[1] Guthaus M R .MiBench: A free, commercially representative embedded benchmark suite[J].Proc. WWC, 2001, 2001.DOI:10.1109/WWC.2001.990739.
[2] https://github.com/riscv-software-src/riscv-tests

1. 研究背景

基准程序集分析

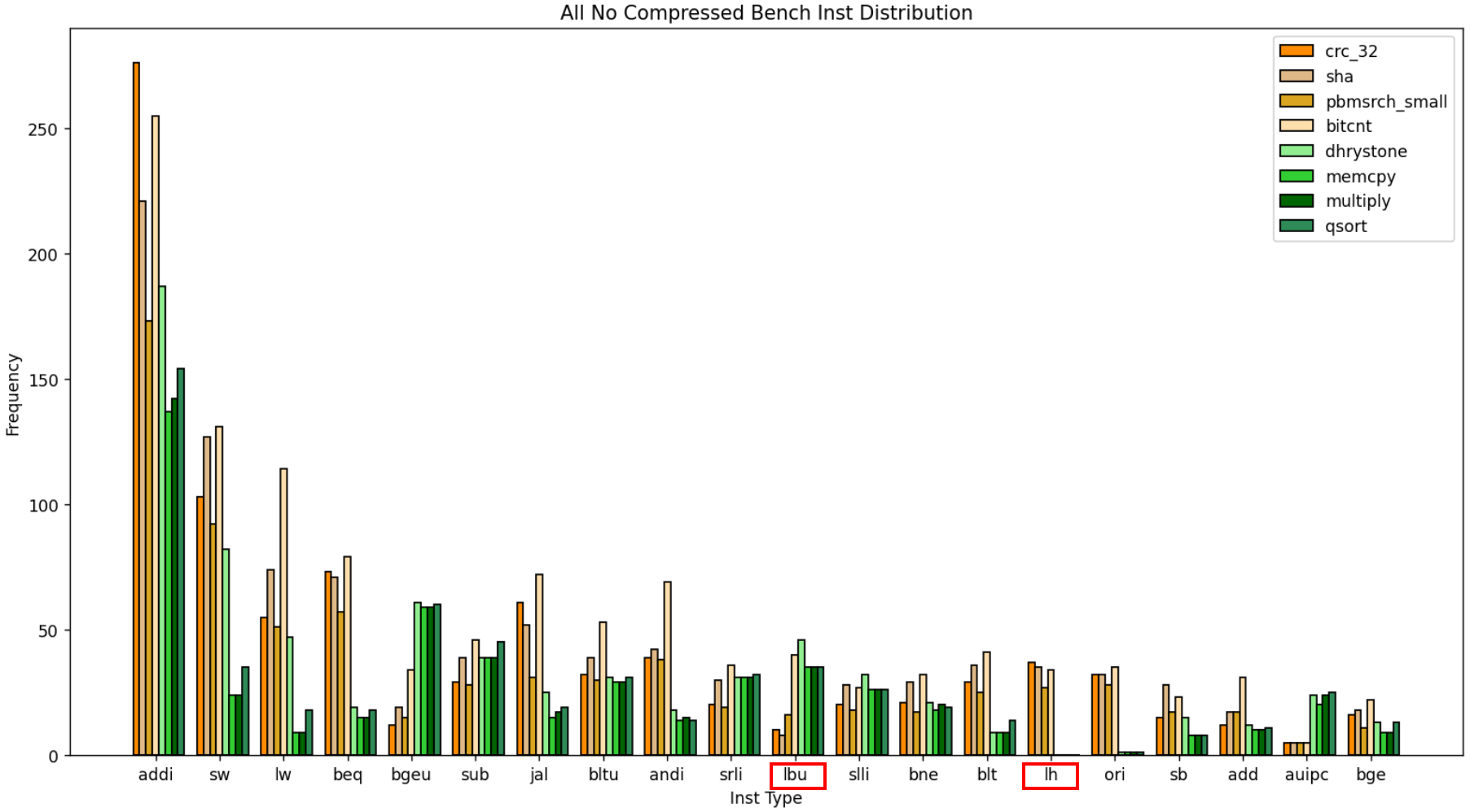
- 条件跳转指令比例较大
- **RV32C**中对于条件指令的支持只有**beqz**和**bnez**指令，对其他条件跳转支持较少



BenchMarks中非压缩指令分布（前**20**条）

1. 研究背景

基准程序集分析



BenchMarks中非压缩指令分布（前20条）

Load/store指令的立即数分布

Load/Store type inst imm analyse

index	InstType	Total	<=6bit	<=6bitRate
0	lbu	225	222	0.987
1	lhu	73	73	1.0
2	lh	133	133	1.0
3	sh	106	104	0.981
4	sb	122	116	0.951
5	lb	0	0	0.0

- **lbu、lh、sh等指令的立即数较小，不大于6bit的比例极高。**

1. 研究背景

基准程序集分析

由原先RV32C中的固有缺陷以及我们对benchmarks的统计分析，我们主要有以下4点定制指令的目标：

- **定制原有压缩指令无法实现的操作**，比如CSR操作、环境调用及sll类型等指令。
- **定制使用原有压缩指令实现非常复杂的操作**，比如无符号加载字节、加载半字等指令。
- **定制使用频率较高的指令**，比如条件跳转中的beq、bgeu等指令。
- **定制长立即数加载指令**，以此来简化长立即数的加载。

2. tinyRV指令集技术

寄存器地址缩减

RV32E只支持16个通用寄存器，因此寄存器地址只需要4bit即可，因此可以将之前RV32C中的5bit寄存器地址缩减到4bit，并且将空闲出来的空间用作**功能码区间**或者**立即数长度**。这种方法主要针对CR、CI和CSS类型进行定制，其中CR类型指令具有两个5bit寄存器地址，因此可以**增加2bit功能码区间**；而CI和CSS类型具有一个5bit的寄存器地址，因此可以**增加1bit的立即数长度或1bit功能码区间**。最终这三种指令格式定制如下：

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CR	funct4				rd/rs1					rs2					op	
CI	funct3			imm	rd/rs1					imm					op	
CSS	funct3			imm						rs2					op	

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CR	funct4				func	rd/rs1				func	rs2				op	
CI	funct3			imm	func/imm	rd/rs1				imm					op	
CSS	funct3			imm						imm	rs2				op	

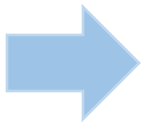
2. tinyRV指令集技术

寄存器地址缩减

[100]-[10]区间具有两个5bit寄存器地址，并且需要依靠[6:2]段的值来进行译码，因此我们将寄存器地址进行缩减，使用多余的2bit空间作为funct码，以此来扩充指令数量并且可以不依靠[6:2]段数据来判断指令类型。此区间包含ebreak指令，因此我们将ecall、mret指令定制到此区间。具体的定制结果如下图所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	INST
100			0	rs1					0					10		C.JR
100			1	rs1					0					10		C.JALR
100			0	rd					rs2					10		C.MV
100			1	rs1/rd					rs2					10		C.ADD
100			1	0					0					10		C.EBREAK

之前的指令格式



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	INST
100			0	0	rs1				0	0				10		C.JR
100			0	1	rs1				0	0				10		C.JALR
100			1	0	rd				0	rs2				10		C.MV
100			1	1	rs1/rd				0	rs2				10		C.ADD
100			0	1	0				1	0				10		C.ECALL
100			1	0	0				1	0				10		C.EBREAK
100			1	1	0				1	0				10		C.MRET

定制之后的指令格式

2. tinyRV指令集技术

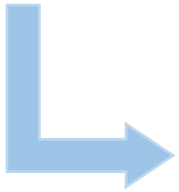
寄存器地址缩减

寄存器地址缩减也可以增强[000]-[01]、[010]-[01]、[011]-[01]、[010]-[10]以及[110]-[10]区间的指令，为这些区间的指令额外增加立即数的长度。具体的定制结果如下。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	INST
010			im[5]	rd					nzuimm[4:2|7:6]					10		C.LWSP
110			uimm[5:2|8:6]						rs2					10		C.SWSP
000			im[5]	0					nzimm[4:0]					01		C.NOP
000			im[5]	rs1/rd					nzimm[4:0]					01		C. ADDI
010			im[5]	rd					imm[4:0]					01		C.LI
011			im[9]	2					nzimm[4|6|8:7|5]					01		C.addi16sp
011			im[17]	rd					imm[16:12]					01		C.LUI

- 对于lwsp和swsp两条指令，均可以增加1bit的立即数空间。
- 对于addi和nop指令，增加了1bit立即数空间，并且区分两条指令时更加简单。

- 对于li、addi16sp和lui指令，增加了1bit的立即数空间，并且lui和addi16sp的区分更加简单。



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	INST
010			im[5]	im[8]	rd				nzuimm[4:2|7:6]					10		C.LWSP
110			uimm[5:2|7:6]						im[8]	rs2				10		C.SWSP
000			im[5]	im[6]	0				nzimm[4:0]					01		C.NOP
000			im[5]	im[6]	rs1/rd				nzimm[4:0]					01		C. ADDI
010			im[5]	im[6]	rd				imm[4:0]					01		C.LI
011			im[9]	im[10]	2				nzimm[4|6|8:7|5]					01		C.addi16sp
011			im[17]	im[18]	rd				imm[16:12]					01		C.LUI

2. tinyRV指令集技术

区间重排

对于移位指令，RV32E中的slli、srli和srai的移位量shamt的长度为5bit，而RV32C中这些指令的立即数长度为6bit，因此可以缩减立即数长度，对于[000]-[10]区间，即c.slli指令区间，还可以缩减寄存器地址，所以此区间可以有多余2bit空间来扩展功能码，故可以将[100]-[01]区间中的srli和srai两条指令并入此区间，并且将sltiu指令定制到此区间。之后对[100]-[01]区间进行进一步的规划，增加寄存器地址位宽或立即数长度，以此来增强指令。最终将这两个区间进行重排之后如下图。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	INST
100			0	11		rs1'/rd'			00		rs2'			01		C.SUB
100			0	11		rs1'/rd'			01		rs2'			01		C.XOR
100			0	11		rs1'/rd'			10		rs2'			01		C.OR
100			0	11		rs1'/rd'			11		rs2'			01		C.AND
100			im[5]	10		rs1'/rd'			imm[4:0]					01		C.ANDI
100			im[5]	00		rs1'/rd'			nzuimm[4:0]					01		C.SRLI
100			im[5]	01		rs1'/rd'			nzuimm[4:0]					01		C.SRAI

000			im[5]	rs1/rd					nzuimm[4:0]					10		C.SLLI

之前的指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	INST
100			00		rs1/rd				0	rs2				01		C.SUB
100			01		rs1/rd				0	rs2				01		C.XOR
100			10		rs1/rd				0	rs2				01		C.OR
100			11		rs1/rd				0	rs2				01		C.AND
100			imm[5:4]		rs1/rd				1	nzuimm[3:0]				01		C.ANDI

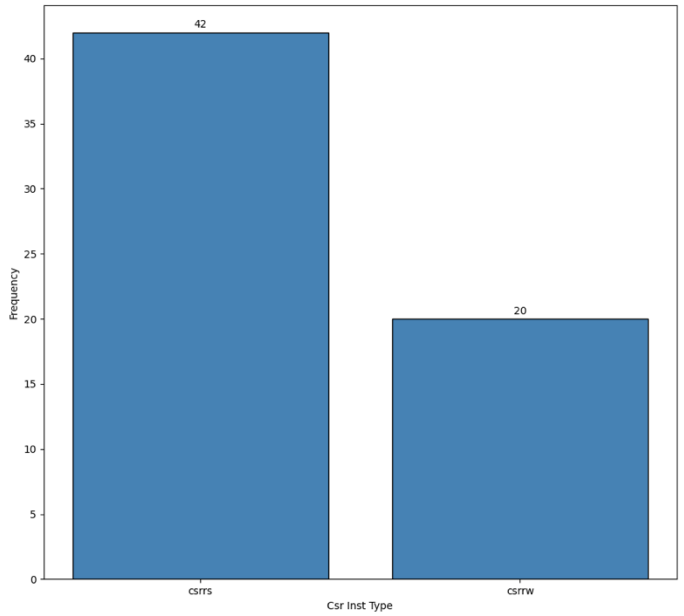
000		00		rs1/rd				nzuimm[4:0]					10		C.SRLI
000		01		rs1/rd				nzuimm[4:0]					10		C.SRAI
000		10		rs1/rd				nzuimm[4:0]					10		C.SLLI
000		11		rs1/rd				nzuimm[4:0]					10		C.SLTIU

定制之后的指令格式

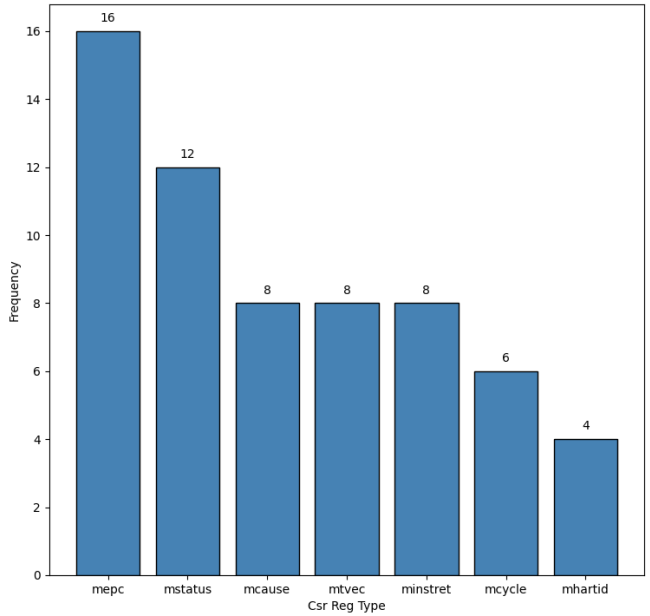
2. tinyRV指令集技术

CSR指令定制

CSR指令分布



CSR寄存器分布



从基准测试集的统计结果可以看出来，CSR类型的指令占比非常少，并且使用到的CSR寄存器不超过8个。因此我们使用一个区间来定制CSR类型中的CSRRW、CSRRC和CSRRS指令，并且给CSR寄存器、源寄存器和目的寄存器均分配3bit的地址空间。最终的CSR类型指令定制如下。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	INST
011			csr[2]	00		rs1			csr[1:0]		rd			00		C.CSRRW
011			csr[2]	01		rs1			csr[1:0]		rd			00		C.CSRRC
011			csr[2]	10		rs1			csr[1:0]		rd			00		C.CSRRS
011			csr[2]	11		rs1			csr[1:0]		rd			00		Reserve

CSR类型指令定制结果

条件跳转指令定制

bgeu和bltu指令数量较多，比例较大，所以考虑进行定制，并且bge和bgeu可以转换为blt和bltu操作，因此只需要定制blt和bltu指令即可。同样，beq和bne指令数量较多，而原先的RV32C中beqz和bnez只能与0进行比较，故定制beq和bne两条指令。具体的条件跳转指令定制如下。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	INST
001			uimm[5:3]			rs1			uimm[2:1]		rs2			00		C.BLT
001			uimm[5:3]			rs1			uimm[2:1]		rs2			10		C.BLTU
101			uimm[5:3]			rs1			uimm[2:1]		rs2			00		C.BEQ
101			uimm[5:3]			rs1			uimm[2:1]		rs2			10		C.BNE

条件跳转类型指令定制结果

Load与Store指令定制

RV32C中有lw、sw以及lwsp、swsp指令来支持对word的存取，没有对于半字、字节的存取操作，而对BenchMarks中的load/store指令分析，其立即数长度较小，不大于6bit的比例极高，故定制时考虑支持6bit的立即数，并且默认源寄存器为SP寄存器。最终定制结果如下。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	INST
100			im[3]	0	rd				nzuimm[2:0|5:4]					00		C.LBUSP
100			im[3]	1	rd				nzuimm[2:0|5:4]					00		C.LBSP
011			im[3]	1	rd				nzuimm[2:0|5:4]					10		C.LHUSP
011			im[3]	0	rd				nzuimm[2:0|5:4]					10		C.LHSP
111			uimm[3:0|5:4]						0	rs2				10		C.SBSP
111			uimm[3:0|5:4]						1	rs2				10		C.SHSP

Load/Store类型指令定制结果

其他指令定制

其他指令主要为RV32C中没有的操作进行定制的指令，比如移位指令（sll、srl及sra）、小于置位指令（slt和sltu）以及auipc指令。这些指令主要是R型指令，因此定制为CR类型，除此之外，多余空间定制为两条寄存器跳转指令。具体的定制结果如下。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	INST
100			0	0	rd/rs1				0	rs2				00		C.SLL
100			0	1	rd/rs1				0	rs2				00		C.SRL
100			1	0	rd/rs1				0	rs2				00		C.SRA
100			1	1	rd				0	imm[15:12]				00		C.AUIPC
100			0	0	rd/rs1				1	rs2				00		C.SLT
100			0	1	rd/rs1				1	rs2				00		C.SLTU
100			1	0	rs1				1	rs2				00		C.BEQZR
100			1	1	rs1				1	rs2				00		C.BNEZR

其他一些指令的定制结果

2. tinyRV指令集技术

长立即数加载指令定制（可选）

假如只需要支持16位指令，可以选择将[11]区间定制为一条长立即数加载指令，此条指令将12位立即数加载到目的寄存器当中。若选择仅将tinyRV当做RV32C的一个扩展，则此区间仍为32位指令区间。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	INST
imm[11:5]							rd		imm[4:0]					11		C.LIW

定制长立即数加载指令

3. tinyRV指令集

tinyRV指令区间

[15:13] / [1:0]	000	001	010	011	100	101	110	111
00	ADDI4SPN	FLD	LW	FLW	Reserve	FSD	SW	FSW
01	ADDI	JAL	LI	LUI/ ADDI16SP	MISC-ALU	J	BEQZ	BNEZ
10	SLLI	FLDSP	LWSP	FLWSP	J[AL]R/MV/ ADD/EBREAK	FSDSP	SWSP	FSWSP

[15:13] / [1:0]	000	001	010	011	100	101	110	111
00	ADDI4SPN	BLT	LW	CSRRW/ CSRRC/ CSRRS	LBUSP\ LBSP	BEQ	SW	SLL/SRL(A)/ SLT[U]/BEQR/ BNER/STOREPC
01	ADDI	JAL	LI	LUI/ ADDI16SP	SUB/XOR/ OR/AND/ANDI	J	BEQZ	BNEZ
10	SLLI/SRAI /SLLI/SLTIU	BLTU	LWSP	LHUSP\ LHSP	J[AL]R/MV/ ADD/EBREAK/ ECALL/MRET	BNE	SWSP	SBSP\ SHSP
11	LIW							

3. tinyRV指令集

tinyRV指令详情

- CR类型
- CJ类型
- CIW类型
- CB类型
- CL类型
- CSS类型

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	INS	Type
100			00		rs1/rd				0	rs2				01		C.SUB	CR
100			01		rs1/rd				0	rs2				01		C.XOR	
100			10		rs1/rd				0	rs2				01		C.OR	
100			11		rs1/rd				0	rs2				01		C.AND	
100			1	0	rd				0	rs2				10		C.MV	
100			1	1	rs1/rd				0	rs2				10		C.ADD	
100			0	0	rd/rs1				0	rs2				00		C.SLL	
100			0	1	rd/rs1				0	rs2				00		C.SRL	
100			1	0	rd/rs1				0	rs2				00		C.SRA	
100			0	0	rd/rs1				1	rs2				00		C.SLT	
100			0	1	rd/rs1				1	rs2				00		C.SLTU	
100			1	0	rd/rs1				1	rs2				00		C.BEQZR	
100			1	1	rd/rs1				1	rs2				00		C.BNEZR	
001			imm[11|4|9:8|10|6|7|3:1|5]											01		C.JAL	CJ
101			imm[11|4|9:8|10|6|7|3:1|5]											01		C.J	
100			0	0	rs1				0	0				10		C.JR	
100			0	1	rs1				0	0				10		C.JALR	
000			nzuimm[5:4|9:6|2|3]									rd'		00		C.addi4spn	CIW
110			imm[8|4:3]			rs1'			imm[7:6|2:1|5]					01		C.beqz	CB
111			imm[8|4:3]			rs1'			imm[7:6|2:1|5]					01		C.bnez	
010			uimm[5:3]			rs1'			uimm[2|6]			rd'		00		C.lw	CL
110			uimm[5:3]			rs1'			uimm[2|6]			rd'		00		C.sw	
010			im[5]	im[8]	rd				nzuimm[4:2|7:6]					10		C.lwsp	CSS
110			uimm[5:2|7:6]						im[8]	rs2				10		C.swsp	

蓝底为完全定制的指令，绿底为在之前的指令格式上进行增强的指令，白底则未改变。

3. tinyRV指令集

tinyRV指令详情

- CI类型

000	im[5]	im[6]		nzimm[4:0]		01	C.nop	CI
000	im[5]	im[6]	rs1/rd	nzimm[4:0]		01	C. addi	
010	im[5]	im[6]	rd	imm[4:0]		01	C.li	
011	im[9]	im[10]	2	nzimm[4|6|8:7|5]		01	C.addi16sp	
011	im[17]	im[18]	rd	imm[16:12]		01	C.lui	
000	00		rs1/rd	nzuimm[4:0]		10	C.SRLI	
000	01		rs1/rd	nzuimm[4:0]		10	C.SRLI	
000	10		rs1/rd	nzuimm[4:0]		10	C.SRAI	
000	11		rs1/rd	nzuimm[4:0]		10	C.SLTIU	
100	imm[5:4]		rs1/rd	1	nzuimm[3:0]	01	C.ANDI	
100	0	1	0	1	0	10	C.ECALL	
100	1	0	0	1	0	10	C.EBREAK	
100	1	1	0	1	0	10	C.MRET	
100	im[3]	0	rd	nzuimm[2:0|5:4]		00	C.LBUSP	
100	im[3]	1	rd	nzuimm[2:0|5:4]		00	C.LBSP	
011	im[3]	1	rd	nzuimm[2:0|5:4]		10	C.LHUSP	
011	im[3]	0	rd	nzuimm[2:0|5:4]		10	C.LHSP	
111	uimm[3:0|5:4]			0	rs2	10	C.SBSP	
111	uimm[3:0|5:4]			1	rs2	10	C.SHSP	
011	csr[2]	00	rs1	csr[1:0]	rd	00	C.CSRRW	
011	csr[2]	01	rs1	csr[1:0]	rd	00	C.CSRRC	
011	csr[2]	10	rs1	csr[1:0]	rd	00	C.CSRRS	
011	csr[2]	11	rs1	csr[1:0]	rd	00	Reserve	
001	uimm[5:3]		rs1	uimm[2:1]	rs2	00	C.BLT	
001	uimm[5:3]		rs1	uimm[2:1]	rs2	10	C.BLTU	
101	uimm[5:3]		rs1	uimm[2:1]	rs2	00	C.BEQ	
101	uimm[5:3]		rs1	uimm[2:1]	rs2	10	C.BNE	
100	1	1	rd	0	imm[15:12]	00	C.AUIPC	
imm[11:5]			rd	imm[4:0]		11	C.LIW	

蓝底为完全定制的指令，绿底为在之前的指令格式上进行增强的指令，白底则未改变。

3. tinyRV指令集

分析与评估

指令类型	指令	转换前需要指令数	转换后需要指令数
U	lui	1/10	1/5
	auipc	3/12	3/7
J	jal	1/15	1/9
R	add sub xor or and	1	1
	sll		1
	slt		1
	sltu		1
	srl		1
	sra		1
S	sb	8/11	1/3
	sh	10/13	1/3
	sw	3/6	1/3
B	beq	2	2/5
	bne	2	2/5
	blt		5
	bge		5
	bltu		5
	bgeu		5

I	addi	1/5	1/2
	xori ori andi	2/5	2
	slli	1	1
	srli	1	1
	srai	1	1
	slti		2
	sltiu		2
	jalr	6/9	5/6
	fence	1/5	1/2
	fence.i		
	lb lh		1/3
	lbu	7/11	3/5
	lhu	13/17	6/9
	lw	2/6	2/3
	ebreak	1	1
	ecall		1
	csrrw		1
	csrrs		1
	csrrc		1
	csrrwi		2
	csrrsi		2
	csrrci		2

4. 总结与展望

总结

我们基于RV32E对RV32C进行了定制工作，通过寄存器地址缩减、区间重排等技术并且对CSR指令、跳转指令以及长立即数加载指令进行定制，得到了一个功能完备的定制压缩指令集tinyRV。对于tinyRV，具有以下两种使用方式：

- **如果追求极致的小，可以采用16位的数据通路，只需要支持完备的tinyRV**
- **如果追求在RV32E下进一步压缩指令缓存，采用32位数据通路，选择RV32E+tinyRV**

展望

针对目前的进展，未来主要有以下两点展望：①增加gcc对tinyRV的编译后端支持；②进一步研究tinyRV对程序压缩率、cache失效率等数据指标的影响。

Thank You!