Discussion 4 – RISC-V

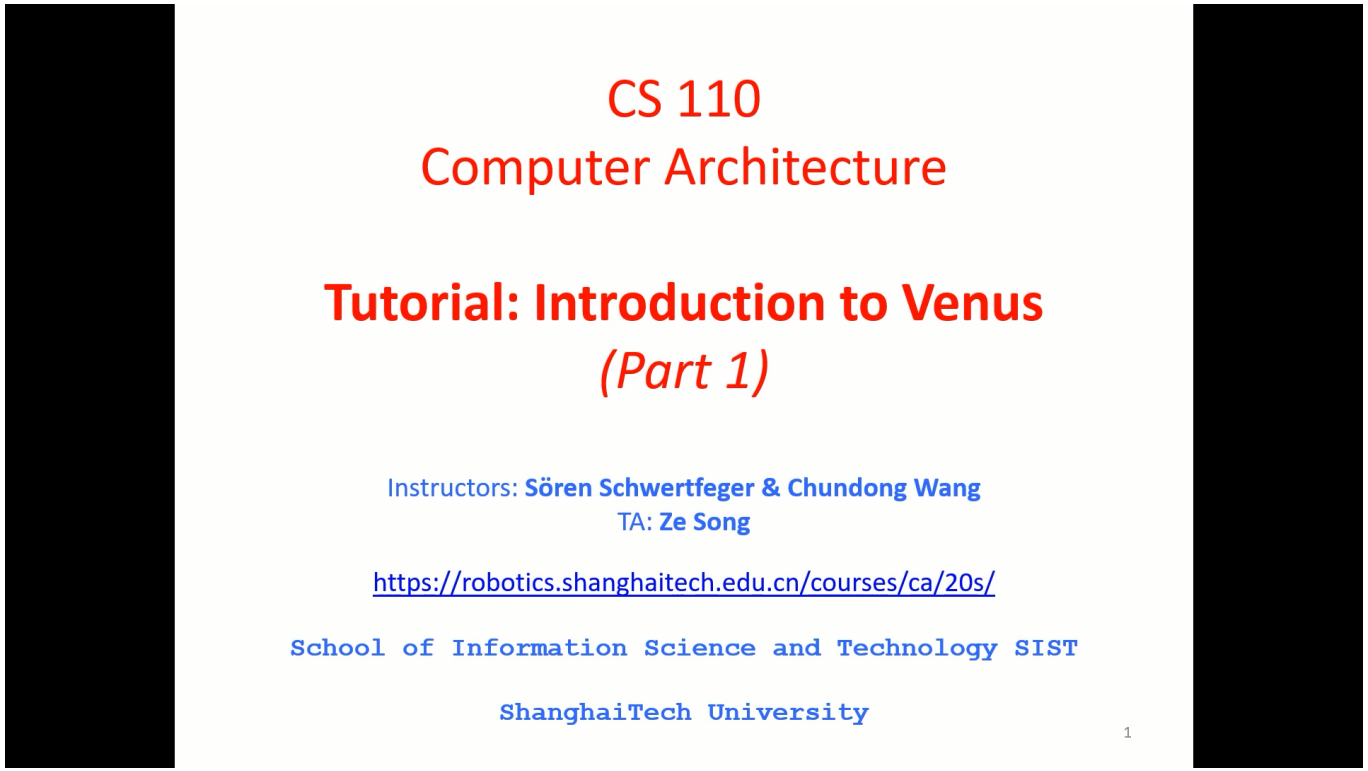
XUBW@SHANGHAITECH.EDU.CN

Agenda

1. Tips for RISC-V programing
2. RISC-V Calling Convention
3. RISC-V Compressed Instruction Set (related to project 1.1/1.2)

Tips for RISC-V programing

Do check Venus tutorial videos provided on the course page



Tips for RISC-V programing

Registers:

- Always use ABI name of registers in assembly codes
- Plan the usage of registers wisely
- Do stick with calling conventions

寄存器	接口名称	描述	在调用中是否保留?
Register	ABI Name	Description	Preserved across call?
x0	zero	Hard-wired zero 硬编码 0	—
x1	ra	Return address 返回地址	No
x2	sp	Stack pointer 栈指针	Yes
x3	gp	Global pointer 全局指针	—
x4	tp	Thread pointer 线程指针	—
x5	t0	Temporary/alternate link register 临时寄存器	No /备用链接寄存器
x6–7	t1–2	Temporaries 临时寄存器	No
x8	s0/fp	Saved register/frame pointer 保存寄存器	Yes /帧指针
x9	s1	Saved register 保存寄存器	Yes
x10–11	a0–1	Function arguments/return values 函数参数	No /返回值
x12–17	a2–7	Function arguments 函数参数	No
x18–27	s2–11	Saved registers 保存寄存器	Yes
x28–31	t3–6	Temporaries 临时寄存器	No

Tips for RISC-V programing

Label:

- Used by Assembler in calculating offset for branch/jump

```
1      addi s0, zero, 0
2      addi s1, zero, 1
3      addi t0, zero, 30
4  loop:
5      beq  s0, t0, exit
6      add  s1, s1, s1
7      addi s0, s0,1
8      jal  x0, loop
9  exit:
```

PC	Machine Code	Basic Code	Original Code
0x0	0x00000413	addi x8 x0 0	addi s0, zero, 0
0x4	0x00100493	addi x9 x0 1	addi s1, zero, 1
0x8	0x01E00293	addi x5 x0 30	addi t0, zero, 30
0xc	0x00540863	beq x8 x5 16	beq s0, t0, exit
0x10	0x009484B3	add x9 x9 x9	add s1, s1, s1
0x14	0x00140413	addi x8 x8 1	addi s0, s0,1
0x18	0xFF5FF06F	jal x0 -12	jal x0, loop

Tips for RISC-V programing

Zero-extended or Sign-extended:

- All about 2's complement

`lb`	I	Load Byte	R[rd] = {56'bM[](7),M[R[rs1]+imm](7:0)}
`lbu`	I	Load Byte Unsigned	R[rd] = {56'b0,M[R[rs1]+imm](7:0)}

- Consider not only how it effects the behavior of an instruction,
- But also how it effects on the input requirement of an instruction.

RISC-V Calling Convention

Caller & Callee

- Caller invoke callee
- Caller should save Caller-Saved registers (to memory) before the call.
- Callee should save Callee-Saved registers at the beginning of its execution and restore them before return.

Steps of function call

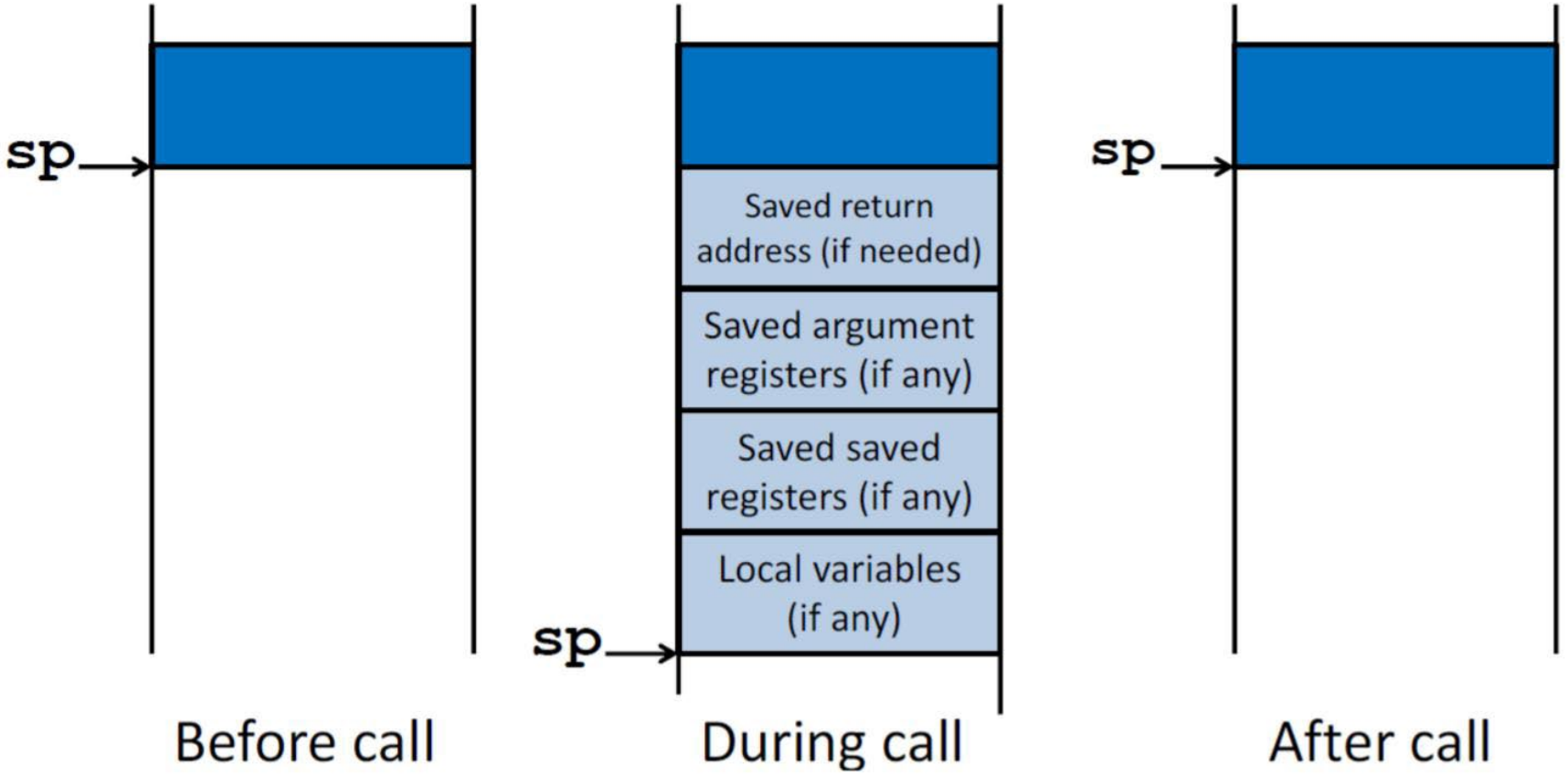
1. Caller put parameters into registers a0-a7.
2. Caller put next line's address into ra and jump to the function label. (using jal)
3. Callee pushes s0-s11, sp onto stack.
4. Callee execution.
5. Callee extract value from stack.
6. Callee jump to ra's address.

RISC-V Calling Convention

REGISTER NAME, USE, CALLING CONVENTION ④

REGISTER	NAME	USE	SAVER
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--
x4	tp	Thread pointer	--
x5-x7	t0-t2	Temporaries	Caller
x8	s0/fp	Saved register/Frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Function arguments/Return values	Caller
x12-x17	a2-a7	Function arguments	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller

RISC-V Calling Convention



RISC-V Calling Convention

Tips:

- If you are not modifying s0-s11, you won't need to save/restore them.
- a0-a7 are your friends, you are welcome to do stuff to the arguments in-place.
- t0-t6 are also your friends.
- s0-s11 are your friends, if you are more of a caller.
- ra, sp, gp, tp are your father, be careful with them.

RISC-V Compressed Instruction Set (Related to project 1.1/1.2)

Length of instruction:
- 16-bit instead of 32-bit

Alignment of PC:
- 2-byte instead of 4-byte

Immediate:
- Smaller

Registers accessible:
- Limited

Format	Meaning	15 14 13	12	11 10	9 8 7	6 5	4 3 2	1 0
CR	Register	funct4		rd/rs1		rs2		op
CI	Immediate	funct3	imm	rd/rs1		imm		op
CSS	Stack-relative Store	funct3	imm			rs2		op
CIW	Wide Immediate	funct3	imm				rd′	op
CL	Load	funct3	imm		rs1′	imm	rd′	op
CS	Store	funct3	imm		rs1′	imm	rs2′	op
CB	Branch	funct3	offset		rs1′	offset		op
CJ	Jump	funct3	jump target					op

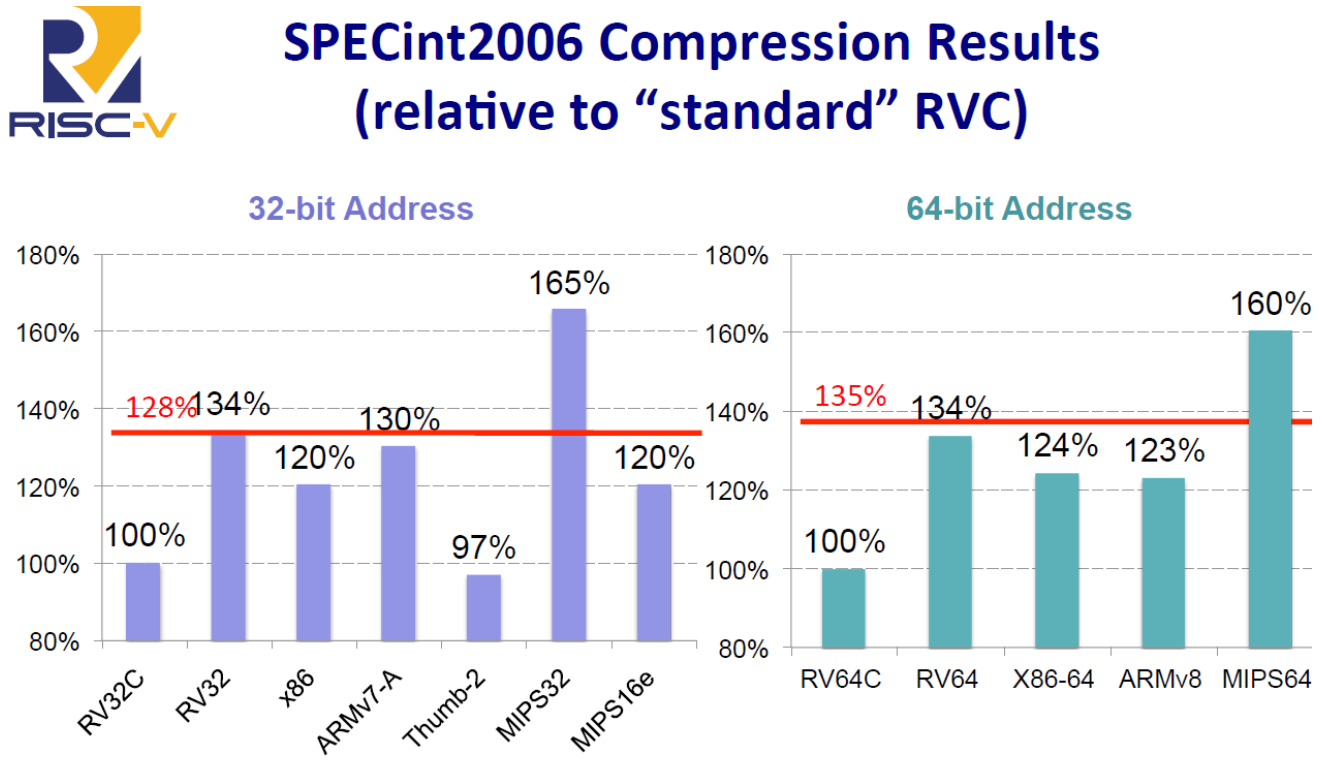
	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

RISC-V Compressed Instruction Set

For what?

- To save static code space for low-end embedded devices, or
- To reduce cache footprint for high-end commercial workloads

Save 25% space compared to standard RV32 code.



- MIPS delayed branch slots increase code size
- RV64C only 64-bit address ISA with 16-bit instructions
- Thumb2 only 32-bit address ISA smaller than RV32C

Credit:
2nd RISC-V Workshop, Berkeley, CA

RISC-V Compressed Instruction Set

RVC uses a simple compression scheme that offers shorter 16-bit versions of common 32-bit RISC-V instructions when:

- the immediate or address offset is small, or
- one of the registers is the zero register (x0), the ABI link register (x1), or the ABI stack pointer (x2), or
- the destination register and the first source register are identical, or
- the registers used are the 8 most popular ones.

The lowest 2 bits of an instruction:

- 11 - not RVC
- 00, 01, 10 - RVC

Integer Register-Register Operations

4	5	5	2
C.MV	dest≠0	src≠0	C0
C.ADD	dest≠0	src≠0	C0

These instructions use the CR format.

C.MV copies the value in register *rs2* into register *rd*. C.MV expands into `add rd, x0, rs2`.

C.ADD adds the values in registers *rd* and *rs2* and writes the result to register *rd*. C.ADD expands into `add rd, rd, rs2`.

RISC-V Compressed Instruction Set

RVC allows 16-bit instructions to be freely intermixed with 32-bit instructions, with the latter now able to start on any 16-bit boundary.

Run | Step | Prev | Reset | Dump | Trace

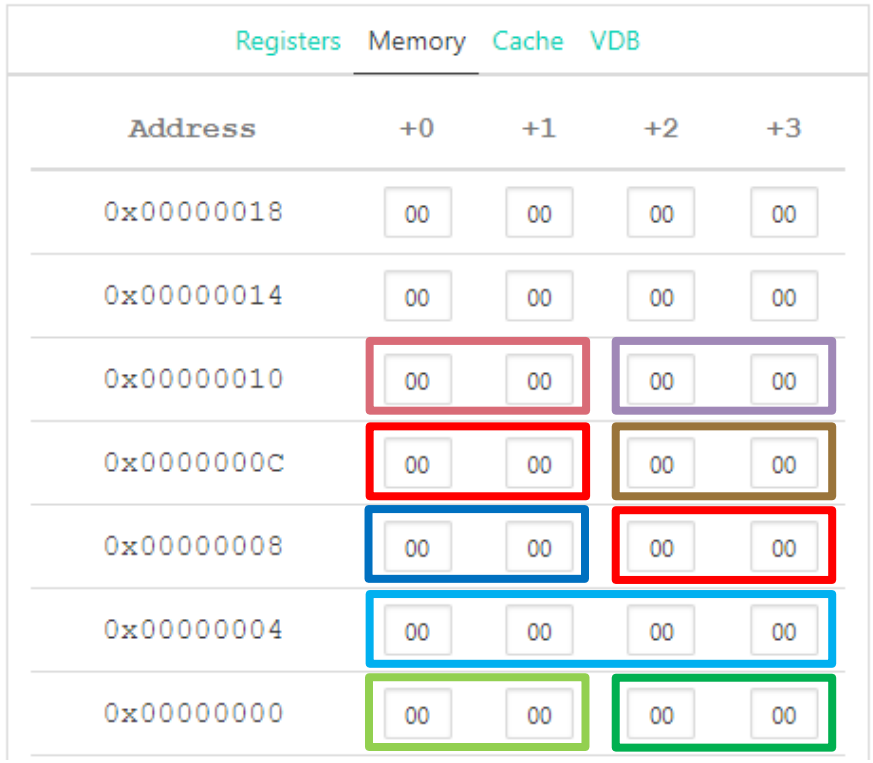
PC	Machine Code	Basic Code	Original Code
0x0	0x00000413	addi x8 x0 0	addi s0, zero, 0
0x4	0x00100493	addi x9 x0 1	addi s1, zero, 1
0x8	0x01E00293	addi x5 x0 30	addi t0, zero, 30
0xc	0x00540863	beq x8 x5 16	beq s0, t0, exit
0x10	0x009484B3	add x9 x9 x9	add s1, s1, s1
0x14	0x00140413	addi x8 x8 1	addi s0, s0,1
0x18	0xFF5FF06F	jal x0 -12	jal x0, loop

Registers | Memory | Cache | VDB

Address	+0	+1	+2	+3
0x00000018	6F	F0	5F	FF
0x00000014	13	04	14	00
0x00000010	B3	84	94	00
0x0000000C	63	08	54	00
0x00000008	93	02	E0	01
0x00000004	93	04	10	00
0x00000000	13	04	00	00

RISC-V Compressed Instruction Set

RVC allows 16-bit instructions to be freely intermixed with 32-bit instructions, with the latter now able to start on any 16-bit boundary.




```
.text
    foo
    bar
    foo  (RV32)
    bar
    foo  (RV32, not 4-byte aligned)
    bar
    foo
    bar
```

Q&A / End of discussion

Thank you for attending the discussion!

Wish you good luck doing homework/projects/exams!