

第7章 指令系统

第1讲 概述与指令系统设计

第2讲 指令系统实例：RISC-V架构

第二讲指令系统实例：RISC-V

主要内容

- ◆ RISC-V指令系统概述
- ◆ RISC-V指令参考卡和指令格式
- ◆ RISC-V基础整数指令集
 - 整数运算
 - 控制转移
 - 存储访问
 - 系统控制
- ◆ RISC-V可选的扩展指令集

RISC-V指令系统概述

◆ 设计目标

- 广泛的适应性：从最袖珍的嵌入式微控制器，到最快的高性能计算机
- 支持各种异构处理架构，成为定制加速器的基础
- 稳定的基础指令集架构，并能灵活扩展，且扩展时不影响基础部分

◆ 开源理念和设计原则

- 本着“指令集应自由（Instruction Set Want to be Free）”的理念，指令集完全公开，且无需为指令集付费
- 由一个非盈利性质的基金会管理，以保持指令集稳定，加快生态建设
- 2020年基金会总部从美国迁到中立国瑞士，坚持开放自由、坚持为全世界服务的理念，被卡脖子的情况大大减少
- 与以前的增量ISA不同，遵循“大道至简”的设计哲学，采用模块化设计，既保持基础指令集的稳定，也保证扩展指令集的灵活配置
- 特点：具有模块化结构，稳定性和可扩展性好，在简洁性、实现成本、功耗、性能和程序代码量等各方面具有显著优势

RISC-V的模块化结构

– **核心：RV32I**

– **标准扩展集：RV32M、RV32F、RV32D、RV32A**

– **32位架构RV32G = RV32IMAFD**

» **其压缩指令集RV32C (指令长度16位)**

– **64位架构RV64G = RV64IMAFD**

» **其压缩指令集RV64C (指令长度16位)**

指令长度
机器字长
通用寄存器长度
定点运算器
处理数据的长度

同一条指令在RV64I和RV32I中都存在时，其具体行为也是不一样的

– **向量计算RV32V和RV64V;**

– **嵌入式RV32E (RV32I的子集, 16个通用寄存器)**

◆ 核心指令集

： 基础整数指令集

RV32I 和 RV64I

◆ 特权指令：
(参考教材
pp196)

◆ 伪指令举例-增加汇编程序可读性

◆ 压缩指令集
: RV32C和RV64C

Base Integer Instructions: RV32I and RV64I					RV Privileged Instructions								
Category	Name	Fmt	RV32I Base		+RV64I		Category	Name	Fmt	RV mnemonic			
Shifts	Shift Left Logical	R	SLL	rd,rs1,rs2	SLLW	rd,rs1,rs2	Trap	Mach-mode trap return	R	MRET			
	Shift Left Log. Imm.	I	SLLI	rd,rs1,shamt	SLLIW	rd,rs1,shamt		Supervisor-mode trap return	R	SRET			
	Shift Right Logical	R	SRL	rd,rs1,rs2	SRLW	rd,rs1,rs2			Interrupt	Wait for Interrupt	R	WFI	
	Shift Right Log. Imm.	I	SRLI	rd,rs1,shamt	SRLIW	rd,rs1,shamt				MMU	Virtual Memory FENCE	R	SFENCE.VMA rs1,rs2
	Shift Right Arithmetic	R	SRA	rd,rs1,rs2	SRAW	rd,rs1,rs2					Examples of the 60 RV Pseudoinstructions		
Shift Right Arith. Imm.	I	SRAI	rd,rs1,shamt	SRAIW	rd,rs1,shamt	Branch = 0 (BEQ rs,x0,imm)	B				BEQZ rs,imm		
Arithmetic	ADD	R	ADD	rd,rs1,rs2	ADDW	rd,rs1,rs2	Jump (uses JAL x0,imm)	J			J imm		
	ADD Immediate	I	ADDI	rd,rs1,imm	ADDIW	rd,rs1,imm	MoVe (uses ADDI rd,rs,0)	R	MV rd,rs				
	SUBtract	R	SUB	rd,rs1,rs2	SUBW	rd,rs1,rs2	RETurn (uses JALR x0,0,ra)	I	RET				
	Load Upper Imm	U	LUI	rd,imm	Optional Compressed (16-bit) Instruction Extension: RV32C								
	Add Upper Imm to PC	U	AUIPC	rd,imm									
Logical	XOR	R	XOR	rd,rs1,rs2	Loads	Load Word	CL	C.LW	rd',rs1',imm	LW	rd',rs1',imm*4		
	XOR Immediate	I	XORI	rd,rs1,imm		Load Word SP	CI	C.LWSP	rd,imm	LW	rd,sp,imm*4		
	OR	R	OR	rd,rs1,rs2		Float Load Word SP	CL	C.FLW	rd',rs1',imm	FLW	rd',rs1',imm*8		
	OR Immediate	I	ORI	rd,rs1,imm		Float Load Word	CI	C.FLWSP	rd,imm	FLW	rd,sp,imm*8		
	AND	R	AND	rd,rs1,rs2		Float Load Double	CL	C.FLD	rd',rs1',imm	FLD	rd',rs1',imm*16		
Compare	Set <	R	SLT	rd,rs1,rs2	Float Load Double SP	CI	C.FLDSP	rd,imm	FLD	rd,sp,imm*16			
	Set < Immediate	I	SLTI	rd,rs1,imm	Stores	Store Word	CS	C.SW	rs1',rs2',imm	SW	rs1',rs2',imm*4		
	Set < Unsigned	R	SLTU	rd,rs1,rs2		Store Word SP	CSS	C.SWSP	rs2,imm	SW	rs2,sp,imm*4		
	Set < Imm Unsigned	I	SLTIU	rd,rs1,imm		Float Store Word	CS	C.FSW	rs1',rs2',imm	FSW	rs1',rs2',imm*8		
	Branches	Branch =	B	BEQ		rs1,rs2,imm	Float Store Word SP	CSS	C.FSWSP	rs2,imm	FSW	rs2,sp,imm*8	
Branch ≠		B	BNE	rs1,rs2,imm		Float Store Double	CS	C.FSD	rs1',rs2',imm	FSD	rs1',rs2',imm*16		
Branch <		B	BLT	rs1,rs2,imm	Float Store Double SP	CSS	C.FSDSP	rs2,imm	FSD	rs2,sp,imm*16			
Branch ≥		B	BGE	rs1,rs2,imm	Arithmetic	ADD	CR	C.ADD	rd,rs1	ADD	rd,rd,rs1		
Branch < Unsigned		B	BLTU	rs1,rs2,imm		ADD Immediate	CI	C.ADDI	rd,imm	ADDI	rd,rd,imm		
Branch ≥ Unsigned	B	BGEU	rs1,rs2,imm	ADD SP Imm * 16		CI	C.ADDI16SP	x0,imm	ADDI	sp,sp,imm*16			
Jump & Link	J&L	J	JAL	rd,imm		ADD SP Imm * 4	CIW	C.ADDI4SPN	rd',imm	ADDI	rd',sp,imm*4		
	Jump & Link Register	I	JALR	rd,rs1,imm		SUB	CR	C.SUB	rd,rs1	SUB	rd,rd,rs1		
	Synch	Synch thread	I	FENCE	Environment	AND	CR	C.AND	rd,rs1	AND	rd,rd,rs1		
		Synch Instr & Data	I	FENCE.I		AND Immediate	CI	C.ANDI	rd,imm	ANDI	rd,rd,imm		
		Environment	CALL	I		ECALL	OR	CR	C.OR	rd,rs1	OR	rd,rd,rs1	
BREAK			I	EBREAK		eXclusive OR	CR	C.XOR	rd,rs1	AND	rd,rd,rs1		
Control Status Register (CSR)			Read/Write	I		CSRRW	rd,csr,rs1	MoVe	CR	C.MV	rd,rs1	ADD	rd,rs1,x0
	Read & Set Bit		I	CSRRS	rd,csr,rs1	Load Immediate	CI	C.LI	rd,imm	ADDI	rd,x0,imm		
	Read & Clear Bit		I	CSRRC	rd,csr,rs1	Load Upper Imm	CI	C.LUI	rd,imm	LUI	rd,imm		
	Read/Write Imm	I	CSRRWI	rd,csr,imm	Shifts	Shift Left Imm	CI	C.SLLI	rd,imm	SLLI	rd,rd,imm		
	Read & Set Bit Imm	I	CSRRSI	rd,csr,imm		Shift Right Ari. Imm.	CI	C.SRAI	rd,imm	SRAI	rd,rd,imm		
Read & Clear Bit Imm	I	CSRRCI	rd,csr,imm	Shift Right Log. Imm.		CI	C.SRLI	rd,imm	SRLI	rd,rd,imm			
Loads	Load Byte	I	LB	rd,rs1,imm		Branches	Branch=0	CB	C.BEQZ	rs1',imm	BEQ	rs1',x0,imm	
	Load Halfword	I	LH	rd,rs1,imm			Branch≠0	CB	C.BNEZ	rs1',imm	BNE	rs1',x0,imm	
	Load Byte Unsigned	I	LBU	rd,rs1,imm	Jump		Jump	CJ	C.J	imm	JAL	x0,imm	
	Load Half Unsigned	I	LHU	rd,rs1,imm			Jump Register	CR	C.JR	rd,rs1	JALR	x0,rs1,0	
	Load Word	I	LW	rd,rs1,imm			Jump & Link	J&L	CJ	C.JAL	imm	JAL	ra,imm
Stores	Store Byte	S	SB	rs1,rs2,imm		Jump & Link Register		CR	C.JALR	rs1	JALR	ra,rs1,0	
	Store Halfword	S	SH	rs1,rs2,imm		System Env. BREAK		CI	C.EBREAK		EBREAK		
	Store Word	S	SW	rs1,rs2,imm	+RV64I			Optional Compressed Extension: RV64C					
	Loads	Load Byte	I	LB	rd,rs1,imm			LWU	rd,rs1,imm	All RV32C (except C.JAL, 4 word loads, 4 word stores) plus:			
		Load Halfword	I	LH	rd,rs1,imm		LD	rd,rs1,imm	ADD Word (C.ADDW)	Load Doubleword (C.LD)			
Load Byte Unsigned		I	LBU	rd,rs1,imm	Stores		ADD Imm. Word (C.ADDIW)	Load Doubleword SP (C.LDSP)					
Load Half Unsigned		I	LHU	rd,rs1,imm		SUBtract Word (C.SUBW)	Store Doubleword (C.SD)						
Load Word		I	LW	rd,rs1,imm		SD	rs1,rs2,imm	Store Doubleword SP (C.SDSP)					

指令参考卡②

◆ 扩展指令集

乘除运算指令集

RVM、原子操作

指令集RVA、浮

点运算指令集

RVF和RVD、向

量操作指令集

RVV

◆ 通用寄存器的

调用约定

32个定点通用寄

存器x0~x31； 32

个浮点寄存器

f0~f31；

Optional Multiply-Divide Instruction Extension: RVM						
Category	Name	Fmt	RV32M (Multiply-Divide)		+RV64M	
Multiply	Multiply	R	MUL	rd,rs1,rs2	MULW	rd,rs1,rs2
	Multiply High	R	MULH	rd,rs1,rs2		
	Multiply High Sign/Uns	R	MULHSU	rd,rs1,rs2		
	Multiply High Uns	R	MULHU	rd,rs1,rs2		
Divide	Divide	R	DIV	rd,rs1,rs2	DIVW	rd,rs1,rs2
	Divide Unsigned	R	DIVU	rd,rs1,rs2		
Remainder	REMAinder	R	REM	rd,rs1,rs2	REMW	rd,rs1,rs2
	REMAinder Unsigned	R	REMU	rd,rs1,rs2	REMUW	rd,rs1,rs2

Optional Atomic Instruction Extension: RVA						
Category	Name	Fmt	RV32A (Atomic)		+RV64A	
Load	Load Reserved	R	LR.W	rd,rs1	LR.D	rd,rs1
Store	Store Conditional	R	SC.W	rd,rs1,rs2	SC.D	rd,rs1,rs2
Swap	SWAP	R	AMOSWAP.W	rd,rs1,rs2	AMOSWAP.D	rd,rs1,rs2
Add	ADD	R	AMOADD.W	rd,rs1,rs2	AMOADD.D	rd,rs1,rs2
Logical	XOR	R	AMOXOR.W	rd,rs1,rs2	AMOXOR.D	rd,rs1,rs2
	AND	R	AMOAND.W	rd,rs1,rs2	AMOAND.D	rd,rs1,rs2
	OR	R	AMOOOR.W	rd,rs1,rs2	AMOOOR.D	rd,rs1,rs2
		R				
Min/Max	MINimum	R	AMOMIN.W	rd,rs1,rs2	AMOMIN.D	rd,rs1,rs2
	MAXimum	R	AMOMAX.W	rd,rs1,rs2	AMOMAX.D	rd,rs1,rs2
	MINimum Unsigned	R	AMOMINU.W	rd,rs1,rs2	AMOMINU.D	rd,rs1,rs2
	MAXimum Unsigned	R	AMOMAXU.W	rd,rs1,rs2	AMOMAXU.D	rd,rs1,rs2

Two Optional Floating-Point Instruction Extensions: RVF & RVD						
Category	Name	Fmt	RV32{F D} (SP,DP Fl. Pt.)		+RV64{F D}	
Move	Move from Integer	R	FMV.W.X	rd,rs1	FMV.D.X	rd,rs1
	Move to Integer	R	FMV.X.W	rd,rs1	FMV.X.D	rd,rs1
Convert	ConVerT from Int	R	FCVT.{S D}.W	rd,rs1	FCVT.{S D}.L	rd,rs1
	ConVerT from Int Unsigned	R	FCVT.{S D}.WU	rd,rs1	FCVT.{S D}.LU	rd,rs1
	ConVerT to Int	R	FCVT.W.{S D}	rd,rs1	FCVT.L.{S D}	rd,rs1
	ConVerT to Int Unsigned	R	FCVT.WU.{S D}	rd,rs1	FCVT.LU.{S D}	rd,rs1

				Calling Convention		
Category	Name	Fmt	RV32{F D} (SP,DP Fl. Pt.)	Register	ABI Name	Saver
Load	Load	I	FL{W,D}	rd,rs1,imm		
Store	Store	S	FS{W,D}	rs1,rs2,imm		
	Arithmetic	R	FADD.{S D}	rd,rs1,rs2	x0	zero
		R	FSUB.{S D}	rd,rs1,rs2	x1	ra
		R	FMUL.{S D}	rd,rs1,rs2	x2	sp
		R	FDIV.{S D}	rd,rs1,rs2	x3	gp
Mul-Add	Square Root	R	FSQRT.{S D}	rd,rs1	x4	tp
	Multiply-ADD	R	FMADD.{S D}	rd,rs1,rs2,rs3	x5-7	t0-2
	Multiply-SUBtract	R	FMSUB.{S D}	rd,rs1,rs2,rs3	x8	s0/fp
	Negative Multiply-SUBtract	R	FNMSUB.{S D}	rd,rs1,rs2,rs3	x9	s1
	Negative Multiply-ADD	R	FNMADD.{S D}	rd,rs1,rs2,rs3	x10-11	a0-1
Sign Inject	SIGN source	R	FSGNJ.{S D}	rd,rs1,rs2	x12-17	a2-7
	Negative SIGN source	R	FSGNJN.{S D}	rd,rs1,rs2	x18-27	s2-11
	Xor SIGN source	R	FSGNJX.{S D}	rd,rs1,rs2	x28-31	t3-t6
		R				
Min/Max	MINimum	R	FMIN.{S D}	rd,rs1,rs2	f0-7	ft0-7
	MAXimum	R	FMAX.{S D}	rd,rs1,rs2	f8-9	fs0-1
Compare	compare Float =	R	FEQ.{S D}	rd,rs1,rs2	f10-11	fa0-1
	compare Float <	R	FLT.{S D}	rd,rs1,rs2	f12-17	fa2-7
	compare Float ≤	R	FLE.{S D}	rd,rs1,rs2	f18-27	fs2-11
Categorize	CLASSify type	R	FCLASS.{S D}	rd,rs1	f28-31	ft8-11
Configure	Read Status	R	FRCSR	rd	zero	Hardwired zero
	Read Rounding Mode	R	FRRM	rd	ra	Return address
	Read Flags	R	FRFLAGS	rd	sp	Stack pointer
	Swap Status Reg	R	FSCSR	rd,rs1	gp	Global pointer
	Swap Rounding Mode	R	FSRM	rd,rs1	tp	Thread pointer
	Swap Flags	R	FSFLAGS	rd,rs1	t0-0,ft0-7	Temporaries
	Swap Rounding Mode Imm	I	FSRMI	rd,imm	s0-11,fs0-11	Saved registers
	Swap Flags Imm	I	FSFLAGSI	rd,imm	a0-7,fa0-7	Function args

Optional Vector Extension: RVV						
	Name	Fmt	RV32V/R64V			
SET Vector Len.	SETVL	R	SETVL	rd,rs1		
	Multiply High REMAinder	R	VMULH	rd,rs1,rs2		
Shift Left Log.	VSL	R	VSL	rd,rs1,rs2		
	Shift Right Log.	R	VSR	rd,rs1,rs2		
Shift R. Arith.	VSR	R	VSR	rd,rs1,rs2		
		R				
Load	Load	I	VLD	rd,rs1,imm		
	Load Strided	R	VLDS	rd,rs1,rs2		
Load indeXed	VLDX	R	VLDX	rd,rs1,rs2		
		R				
Store	VST	S	VST	rd,rs1,imm		
	Store Strided	R	VSTS	rd,rs1,rs2		
Store indeXed	VSTX	R	VSTX	rd,rs1,rs2		
		R				
AMO SWAP	AMOSWAP	R	AMOSWAP	rd,rs1,rs2		
	AMO ADD	R	AMOADD	rd,rs1,rs2		
AMO XOR	AMOXOR	R	AMOXOR	rd,rs1,rs2		
	AMO AND	R	AMOAND	rd,rs1,rs2		
AMO OR	AMOOOR	R	AMOOOR	rd,rs1,rs2		
		R				
AMO MINimum	AMOMIN	R	AMOMIN	rd,rs1,rs2		
	AMO MAXimum	R	AMOMAX	rd,rs1,rs2		
Predicate =	VPEQ	R	VPEQ	rd,rs1,rs2		
	Predicate ≠	R	VPNE	rd,rs1,rs2		
Predicate <	VPLT	R	VPLT	rd,rs1,rs2		
	Predicate ≥	R	VPGE	rd,rs1,rs2		
Predicate AND	VPAND	R	VPAND	rd,rs1,rs2		
	Pred. AND NOT	R	VPANDN	rd,rs1,rs2		
Predicate OR	VPOR	R	VPOR	rd,rs1,rs2		
	Predicate XOR	R	VPXOR	rd,rs1,rs2		
Predicate NOT	VPNOT	R	VPNOT	rd,rs1		
	Pred. SWAP	R	VPSWAP	rd,rs1		
MOVE	VMOV	R	VMOV	rd,rs1		
		R				
ConVerT	VCVT	R	VCVT	rd,rs1		
		R				
ADD	VADD	R	VADD	rd,rs1,rs2		
	SUBtract	R	VSUB	rd,rs1,rs2		
MULTiply	VMUL	R	VMUL	rd,rs1,rs2		
	DIVide	R	VDIV	rd,rs1,rs2		
Square Root	VSQRT	R	VSQRT	rd,rs1,rs2		
		R				
Multiply-ADD	VFMADD	R	VFMADD	rd,rs1,rs2,rs3		
	Multiply-SUB	R	VFMSUB	rd,rs1,rs2,rs3		
Neg. Mul.-SUB	VFNMSUB	R	VFNMSUB	rd,rs1,rs2,rs3		
	Neg. Mul.-ADD	R	VFNMADD	rd,rs1,rs2,rs3		
SIGN inject	VSGNJ	R	VSGNJ	rd,rs1,rs2		
	Neg SIGN inject	R	VSGNJN	rd,rs1,rs2		
Xor SIGN inject	VSGNJX	R	VSGNJX	rd,rs1,rs2		
		R				
MINimum	VMIN	R	VMIN	rd,rs1,rs2		
	MAXimum	R	VMAX	rd,rs1,rs2		
XOR	VXOR	R	VXOR	rd,rs1,rs2		
	OR	R	VOR	rd,rs1,rs2		
AND	VAND	R	VAND	rd,rs1,rs2		
		R				
CLASS	VCLASS	R	VCLASS	rd,rs1		
		R				
SET Data Conf.	VSETDCFG	R	VSETDCFG	rd,rs1		
		R				
EXTRACT	VEXTRACT	R	VEXTRACT	rd,rs1,rs2		
	MERGE	R	VMERGE	rd,rs1,rs2		
SELECT	VSELECT	R	VSELECT	rd,rs1,rs2		
		R				

定点通用寄存器的功能定义和两种汇编表示

寄存器	ABI 名	功能描述	被调用过程保存?
x0	zero	硬编码 0	—
x1	ra	返回地址	否
x2	sp	栈指针	是
x3	gp	全局指针	—
x4	tp	线程指针	—
x5	t0	临时寄存器	否
x6~x7	t1~t2	临时寄存器	否
x8	s0/fp	保存寄存器/帧指针	是
x9	s1	保存寄存器	是
x10~x11	a0~a1	过程参数/返回值	否
x12~x17	a2~a7	过程参数	否
x18~x27	s2~s11	保存寄存器	是
x28~x31	t3~t6	临时寄存器	否

指令长度为32位的RISC-V指令格式

◆ 共有6种指令格式

R-型为寄存器操作数指令

I-型为短立即数或装入 (Load) 指令

S-型为存储 (Store) 指令

B-型为条件跳转指令

U-型为长立即数操作指令

J-型为无条件跳转指令

◆ **opcode**: 7位操作码字段

◆ **rd**、**rs1**和**rs2**: 通用寄存器编号

◆ **imm**: 立即数, 其位数在括号[]中表示

◆ **funct3**和**funct7**: 分别表示3位功能码和7位功能码, 和opcode字段一起定义指令的操作功能

	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		
B	imm[12 10:5]				rs2			rs1			funct3		imm[4:1 11]		opcode		
U	imm[31:12]											rd		opcode			
J	imm[20 10:1 11 19:12]											rd		opcode			

指令长度为16位的RISC-V压缩指令格式

- ◆ 共有8种指令格式。与32位指令相比，16位指令中的一部分寄存器编号还是占5位。指令变短了，但还是**32位架构**，处理的还是32位数据，还是有32个通用寄存器。
- ◆ 为了缩短指令长度，操作码op、功能码funct、立即数imm和另一部分寄存器编号的位数都减少了。
- ◆ 每条16位指令都有功能完全相同的32位指令，在执行时由硬件先转换为32位指令再执行。**目的是：缩短程序代码量，用少量时间换空间！**

	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			
CIW	funct3		imm								rd'		op			
CL	funct3		imm			rs1'			imm		rd'		op			
CS	funct3		imm			rs1'			imm		rs2'		op			
CB	funct3		offset			rs1'			offset				op			
CJ	funct3		jump target										op			

RISC-V基础整数指令集 (RV32I)

◆ 包含:

- 移位 (Shifts)
 - 算术运算 (Arithmetic)
 - 逻辑运算 (Logical)
 - 比较 (Compare)
- } 整数运算类指令
- 分支 (Branch)
 - 跳转链接 (Jump & Link)
- } 控制转移类指令
- 同步 (Synch)
 - 环境 (Environment)
 - 控制状态寄存器 (Control Status Register)
- } 系统控制类指令
- 取数 (Load)
 - 存数 (Store)
- } 存储访问类指令

RTL规定:

R[r]: 通用寄存器r的内容

M[addr]: 存储单元addr的内容

M[R[r]]: 寄存器r的内容所指存储单元的内容

PC: PC的内容

M[PC]: PC所指存储单元的内容

SEXT[imm]: 对imm进行符号扩展

ZEXT[imm]: 对imm进行零扩展

传送方向用←表示, 即传送源在右, 传送目的在左

RISC-V基础整数指令集（RV32I）

整数运算类指令

31		25 24		20 19		15 14		12 11		7 6		0	
imm[31:12]						rd		0110111		U lui			
imm[31:12]						rd		0010111		U auipc			
imm[11:0]				rs1		000		rd		0010011		I addi	
imm[11:0]				rs1		010		rd		0010011		I slti	
imm[11:0]				rs1		011		rd		0010011		I sltiu	
imm[11:0]				rs1		100		rd		0010011		I xori	
imm[11:0]				rs1		110		rd		0010011		I ori	
imm[11:0]				rs1		111		rd		0010011		I andi	
0000000		shamt		rs1		001		rd		0010011		I slli	
0000000		shamt		rs1		101		rd		0010011		I srli	
0100000		shamt		rs1		101		rd		0010011		I srai	
0000000		rs2		rs1		000		rd		0110011		R add	
0100000		rs2		rs1		000		rd		0110011		R sub	
0000000		rs2		rs1		001		rd		0110011		R sll	
0000000		rs2		rs1		010		rd		0110011		R slt	
0000000		rs2		rs1		011		rd		0110011		R sltu	
0000000		rs2		rs1		100		rd		0110011		R xor	
0000000		rs2		rs1		101		rd		0110011		R srl	
0100000		rs2		rs1		101		rd		0110011		R sra	
0000000		rs2		rs1		110		rd		0110011		R or	
0000000		rs2		rs1		111		rd		0110011		R and	

RISC-V基础整数指令集 (RV32I)

U型指令共2条

imm[31:12]	rd	0110111	U lui
imm[31:12]	rd	0010111	U auipc

lui rd, imm20: 将立即数imm20存到rd寄存器高20位, 低12位为0。该指令和 “addi rd, rs1, imm12” 结合, 可以实现对一个32位变量赋初值。

imm[11:0]	rs1	000	rd	0010011	I addi
-----------	-----	-----	----	---------	--------

举例: 请给出C语句 “int x=-8191;” 对应的RISC-V机器级代码 (就是编译)

解: 对应的RISC-V机器指令和汇编指令为:

寻址方式?

1111 1111 1111 1111 1110 00101 0110111 lui x5, 1048574 #R[x5]←
0000 0000 0001 00101 000 00101 0010011 addi x5, x5, 1 #R[x5]←R

立即
寄存器直接

SEXT表示符号扩展

-8191的机器数为: 1111 1111 1111 1111 1110 0000 0000 0001

auipc rd, imm20: 将立即数imm20加到PC (32位) 的高20位上, 结果存rd
可用指令 “auipc x10, 0” 获取当前PC的内容, 存入寄存器x10中。

RISC-V基础整数指令集（RV32I）

I 型指令共9条，其中三条为用立即数指定所移位数的移位指令

31	25 24	20 19	15 14	12 11	7 6	0	
imm[11:0]		rs1	000	rd	0010011		I addi
imm[11:0]		rs1	010	rd	0010011		I slti
imm[11:0]		rs1	011	rd	0010011		I sltiu
imm[11:0]		rs1	100	rd	0010011		I xori
imm[11:0]		rs1	110	rd	0010011		I ori
imm[11:0]		rs1	111	rd	0010011		I andi
0000000	shamt	rs1	001	rd	0010011		I slli
0000000	shamt	rs1	101	rd	0010011		I srli
0100000	shamt	rs1	101	rd	0010011		I srai

操作码opcode：都是0010011，其功能由funct3指定，而当funct3=101时，再有高7位区分是算术右移（srai）还是逻辑右移（srli）。

imm[11:0]：12位立即数，**符号扩展为32位**，作为第2个源操作数，和R[rs1]（寄存器rs1中的内容）进行运算，结果存rd。

shamt：指出移位位数，因为最多移31位，故用5位即可。

RISC-V基础整数指令集（RV32I）

举例：请给出C语句 “int x=8191;” 对应的RISC-V机器级代码。

解：8191的机器数为：0000 0000 0000 0000 0001 1111 1111 1111

“lui rd, imm20” 和 “addi rd, rs1, imm12” 结合。如下，对不对？

0000 0000 0000 0000 0001 00101 0110111 lui x5, 1 #R[x5]← 0000 1000H (4096)

1111 1111 1111 00101 000 00101 0010011 addi x5, x5, -1 #R[x5]←R[x5]+SEXT[FFFFH]

不对！因为低12位中第一位为1，addi按**符号扩展**相加！结果为4095。

【注意：机器指令 转换成 汇编形式时，都是按该指令的设计规定来完成的】

可利用addi符号扩展特性进行调整！因为 imm12范围为-2048~2047，故可用lui先装入一个距离目标常数小于2048的数，再通过 addi 进行 加 或 减 (imm12为负时) 来调整！

这里 8191=8192-1，故可先装入8192，再用 addi 减1（加全1）！

0000 0000 0000 0000 0010 00101 0110111 lui x5, 2 #R[x5]← 0000 2000H (8192)

1111 1111 1111 00101 000 00101 0010011 addi x5, x5, -1 #R[x5]←R[x5]+SEXT[FFFFH]

RISC-V基础整数指令集（RV32I）

R型指令共10条

31	25 24	20 19	15 14	12 11	7 6	0	
0000000	rs2	rs1	000	rd	0110011		R add
0100000	rs2	rs1	000	rd	0110011		R sub
0000000	rs2	rs1	001	rd	0110011		R sll
0000000	rs2	rs1	010	rd	0110011		R slt
0000000	rs2	rs1	011	rd	0110011		R sltu
0000000	rs2	rs1	100	rd	0110011		R xor
0000000	rs2	rs1	101	rd	0110011		R srl
0100000	rs2	rs1	101	rd	0110011		R sra
0000000	rs2	rs1	110	rd	0110011		R or
0000000	rs2	rs1	111	rd	0110011		R and

操作码opcode：都是0110011，其功能由funct3指定，而当funct3=000、101时，再由funct7区分是加（add）还是减（sub）、逻辑右移（srl）还是算术右移（sra）。

rs1、rs2、rd：5位通用寄存编号，共32个；两个源操作数分别在rs1和rs2寄存器中，结果存rd。

sll：逻辑左移指令，无算术左移指令。因逻辑左移和算术左移结果完全相同

RISC-V基础整数指令集（RV32I）

4条比较指令：带符号小于（slt、slti）、无符号小于（sltu、sltiu）

例如，“sltiu rd, rs1, imm12” 功能为：将rs1内容与imm12符号扩展结果按无符号整数比较，若小于，则1存入rd中；否则，0存入rd中。

imm[11:0]	rs1	010	rd	0010011	I slti
imm[11:0]	rs1	011	rd	0010011	I sltiu

0000000	rs2	rs1	010	rd	0110011	R slt
0000000	rs2	rs1	011	rd	0110011	R sltu

RISC-V基础整数指令集 (RV32I)

举例：假定变量x、y和z都是long long型，占64位，
x的高、低32位分别存放在寄存器x13、x12中；
y的高、低32位分别存放在寄存器x15、x14中；
z的高、低32位分别存放在寄存器x11、x10中

请写出C语句

“z=x+y;” 对应的
32位字长RISC-V机
器级代码。

解：可通过sltu指令将低32位的进位加入到高32位中。

低32位不涉及符号位，所以使用sltu。用add和addu都可以。

0000000 01110 01100 000 01010 0110011 add x10,x12,x14 #R[x10]←R[x12]+R[x14]

0000000 01100 01010 011 01011 0110011 sltu x11,x10,x12 #若R[x10]<R[x12]，则

使用了临时寄存器x16

R[x11]←1 (若和比加数小，则一定有进位)

0000000 01111 01101 000 10000 0110011 add x16,x13,x15 #R[x16]←R[x13]+R[x15]

0000000 10000 01011 000 01011 0110011 add x11,x11,x16 #R[x11]←R[x11]+R[x16]

0000000	rs2	rs1	000	rd	0110011	R add
0000000	rs2	rs1	011	rd	0110011	R sltu

RISC-V基础整数指令集 (RV32I)

控制转移类指令

31	25 24	20 19	15 14	12 11	7 6	0	
imm[20 10:1 11 19:12]					rd	1101111	J jal
imm[11:0]			rs1	000	rd	1100111	I jalr
imm[12 10:5]	rs2	rs1	000	imm[4:1 11]	1100011		B beq
imm[12 10:5]	rs2	rs1	001	imm[4:1 11]	1100011		B bne
imm[12 10:5]	rs2	rs1	100	imm[4:1 11]	1100011		B blt
imm[12 10:5]	rs2	rs1	101	imm[4:1 11]	1100011		B bge
imm[12 10:5]	rs2	rs1	110	imm[4:1 11]	1100011		B bltu
imm[12 10:5]	rs2	rs1	111	imm[4:1 11]	1100011		B bgeu

J 型: jal 功能为: $PC \leftarrow PC + \text{SEXT}[\text{imm}[20:1] \ll 1]$; $R[\text{rd}] \leftarrow PC + 4$

jal x1/rd/,imm 实现过程调用; **jal x0/rd/,imm** 实现无条件跳转。

I 型: jalr 功能为: $PC \leftarrow R[\text{rs1}] + \text{SEXT}[\text{imm}[12]]$; $R[\text{rd}] \leftarrow PC + 4$

指令 **jalr x0/rd/,x1/rs1/,0** 可实现过程调用的返回。

B型: 皆为分支指令, 其中, bltu、bgeu分别为无符号数比较小于、大于等于转移。转移目标地址 = $PC + \text{SEXT}[\text{imm}[12:1] \ll 1]$

<<1: 指令地址总是2的倍数 (RV32G、RV32C指令分别为4、2字节长)

RISC-V基础整数指令集 (RV32I)

举例：若int型变量x、y、z分别存放在寄存器x5、x6、x7中，写出C语句“z=x+y;”对应的RISC-V机器级代码，要求检测是否溢出。

解：当x、y为int类型时，若“ $y < 0$ 且 $x+y \geq x$ ”或者“ $y \geq 0$ 且 $x+y < x$ ”，则x+y溢出。可通过slti指令对y与0进行比较。

0000000 00110 00101 000 00111 0110011 add x7,x5,x6 #R[x7]←R[x5]+R[x6]

0000 0000 0000 00110 010 11100 0010011 slti x28,x6,0 #若R[x6]<0，则R[x28]←1

0000000 00101 00111 010 11101 0110011 slt x29,x7,x5 #若R[x7]<R[x5] 则R[x29]←1

0000010 11101 11100 001 10000 1100011 bne x28,x29,overflow #若R[x28]≠R[x29]

.....

#则转溢出处理

overflow: xxxxxxxx (某指令)

bne：上面这条指令的二进制编码是怎么确定的？

imm[12 10:5]	rs2	rs1	001	imm[4:1 11]	1100011
--------------	-----	-----	-----	-------------	---------

RISC-V基础整数指令集（RV32I）

0000010 **11101** **11100** **001** 10000 1100011 bne x28,x29,overflow #若R[x28]≠R[x29]

.....

#则转溢出处理

overflow: xxxxxxxx (某指令)

bne:

imm[12 10:5]	rs2	rs1	001	imm[4:1 11]	1100011
--------------	-----	-----	-----	-------------	---------

假定标号为overflow的指令地址与“bne x28...”这条指令地址相距**80字节**，
则“bne x28....”指令中的**偏移量应为80**，
因此，指令中的**立即数为40** —— **40=0000 0010 1000B**，

按照B-型格式对立即数重组，

所以，该指令的机器码为“**0000010** 11101 11100 001 **10000** 1100011”

RISC-V基础整数指令集（RV32I）

存储访问指令

31	25 24	20 19	15 14	12 11	7 6	0	
imm[11:0]		rs1	000	rd	0000011		I lb
imm[11:0]		rs1	001	rd	0000011		I lh
imm[11:0]		rs1	010	rd	0000011		I lw
imm[11:0]		rs1	100	rd	0000011		I lbu
imm[11:0]		rs1	101	rd	0000011		I lhu
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011		S sb
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011		S sh
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011		S sw

I 型：5条取数（Load）指令。功能: $R[rd] \leftarrow M[R[rs1] + SEXT[imm[12]]]$

lbu、lhu：分别为无符号字节、半字取，取出数据按0扩展为32位，装入rd

S型：3条存数（Store）指令。功能: $M[R[rs1] + SEXT[imm[12]]] \leftarrow R[rs2]$

sb、sh：分别将rs2寄存器中低8、低16位写入存储单元中。

汇编形式如： `lw rd, imm12(rs1), sw rs2, imm12(rs1)`

RISC-V基础整数指令集（RV32I）

系统控制类指令

31	25 24	20 19	15 14	12 11	7 6	0	
0000	pred	succ	00000	000	00000	0001111	I fence
0000	0000	0000	00000	001	00000	0001111	I fence.i
000000000000			00000	000	00000	1110011	I ecall
000000000000			00000	000	00000	1110011	I ebreak
csr			rs1	001	rd	1110011	I csrrw
csr			rs1	010	rd	1110011	I csrrs
csr			rs1	011	rd	1110011	I csrrc
csr			zimm	101	rd	1110011	I csrrwi
csr			zimm	110	rd	1110011	I csrrsi
csr			zimm	111	rd	1110011	I csrrci

- fence: RISC-V架构在不同硬件线程之间使用宽松一致性模型，fence和fence.i 两条屏障指令，用于保证一定的存储访问顺序。
- ecall和ebreak: 陷阱（trap）指令，也称自陷指令，主要用于从用户程序陷入到操作系统内核（ecall）或调试环境（ebreak）执行，因此也称为环境（Environment）类指令。
- csrxxx: 6条csr指令用于设置和读取相应的控制状态寄存器（CSR）。

RISC-V可选的扩展指令集

◆ 标准扩展指令集

- RV32I基础指令集之上，可标准扩展RV32M、RV32F/D、RV32A，以形成32位架构合集**RV32IMAFD**，也称为**RV32G**
- RV32G基础上，对每个指令集进行调整和添加，可形成64位架构**RV64G**，原先在RV32G中处理的数据将调整为64位。但为了支持32位数据操作，每个64位架构指令集中都会添加少量32位数据处理指令。

◆ RISC-V扩展集包括

- 针对**64位架构**需要，在**47条RV32I指令基础上**，增加12条整数指令（+RV64I），包括6条32位移位指令、3条32位加减运算指令、两条64位装入（Load）指令和1条64位存储（Store）指令，故RV64I共59条指令。
- 针对乘除运算需要，提供了32位架构乘除运算指令集RV32M中的8条指令，并在此基础上增加了4条**RV64M专用指令**（+RV64M）
- 针对浮点数运算的需要，提供了32位架构的单精度浮点处理指令集RV32F和双精度浮点处理指令集RV32D，并在此基础上分别增加了**RV64F和RV64D专用指令集**（+RV64F）和（+RV64D）。
- 针对事务处理和操作原子性的需要，提供了32位架构原子操作指令集RV32A以及RV64A专用指令集（+RV64A）。关于事务处理和原子性操作问题的说明可参考第8章。

◆ 向量处理指令集RVV、未来可选扩展指令集RVB、RVE、RVH、……

64位架构指令举例

例：在64位RISC-V架构中，如何实现将一个32位常数

00000000 00111101 00000101 00000000装入64位寄存器a0中？

注：在64位架构中，lui指令和32位架构中类似。将一个常数的高20位装入到64位寄存器中，具体规定如下

解：lui指令将常数中的第31~12位0000 0000 0011 1101 0000 (976) 装入到a0寄存器的第31~12位，同时，a0寄存器的第11~0位为全0，高32位按**符号扩展**（第31位为符号）为全0。

再将常数的低12位0101 0000 0000（对应十进制数1280）加到a0中。

因此，实现上述功能对应的汇编指令序列为：

lui a0, 976

addi a0, a0, 1280

回顾：RISC-V中整数的乘、除运算处理

◆ 乘法指令: mul, mulh, mulhu, mulhsu

- mul rd, rs1, rs2: 将低32位乘积存入结果寄存器rd
- Mulh: 将两个乘数同时按带符号整数相乘, 高32位乘积存入rd中
- mulhu: 将两个乘数同时按无符号整数相乘, 高32位乘积存入rd中
- mulhsu: 将两个乘数分别作为带符和无符整数相乘, 高32位乘积存入rd
- 得到64位乘积需要两条连续的指令, 其中一定有一条是mul指令, 硬件实际执行时其实只是执行了一条指令
- 两种乘法指令都不检测溢出, 而是直接把结果写入结果寄存器。由软件根据结果寄存器的值自行判断和处理溢出

◆ 除法指令: div, divu, rem, remu

- div / rem: 按带符号整数做除法, 得到商 / 余数
- divu / remu: 按无符号整数做除法, 得到商 / 余数

◆ RISC-V指令不检测和发出异常 (除0), 而是由系统软件自行处理

◆ 乘法指令 (硬件) 也可以生成溢出标志, 只是RISCV没有这样做。

进一步思考（续第6章内容，请按需回顾）

- ◆ 可以根据高位乘积寄存器和低位乘积寄存器的内容来进行溢出判断（编译器可以生成相关的判断指令，由指令计算并存放溢出标志）

在字长为32位的计算机上，某C函数原型声明为：int imul_overflow(int x, int y); 该函数用于对两个int型变量x和y的乘积（也是int类型）判断是否溢出，若溢出则返回非0，否则返回0。请完成下列任务或回答下列问题。

(2) 已知入口参数x、y分别在寄存器a0、a1中，返回值在a0中，写出实现imul_overflow函数功能的RISC-V汇编指令序列，并给出注解。（编译器中判断溢出的代码）

实现该功能的汇编指令序列不唯一，可能如下——

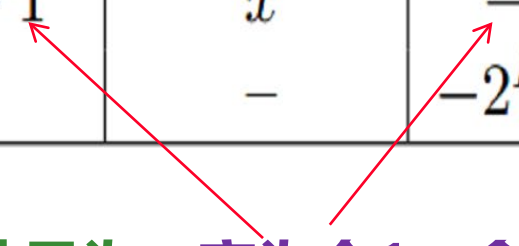
mul	t0, a0, a1	# x*y的低32位在t0中
mulh	a0, a0, a1	# x*y的高32位在a0中
srai	t0, t0, 31	# 乘积的低32位算术右移31位
xor	a0, a0, t0	# 按位异或，若结果为0，表示不溢出

（本处按题意，返回值写入a0，而不是写入溢出标志位）

进一步思考：除法结果的处理

- RISC-V指令不检测和发出异常，而是由系统软件自行处理
如除法错，不触发异常，而用特殊的商和余数来表示

Condition	DIVU[W]	REMU[W]	DIV[W]	REM[W]
Division by zero	$2^L - 1$	x	-1	x
Overflow (signed only)	$-$	$-$	-2^{L-1}	0



若整数 x 除以0，则指令执行结果为：商为全1，余数为 x 。

当最小的负整数除以-1时，会发生结果溢出，此时，相应指令执行结果为：商为被除数（即最小负整数），余数为0。

这样做的好处是：简化流水线的硬件实现

若编译器对除法错进行处理，可查看商和余数来判断

若编译器不处理除法错，则程序就得到错误结果，这种情况下需要程序员进行相应处理

本章总结1

◆ 指令格式

- 定长指令字：所有指令长度一致
- 变长指令字：指令长度有长有短

◆ 操作类型

- 数据传送：数据在寄存器、主存单元、栈顶等处进行传送
- 操作运算：各种算术运算、逻辑运算
- 字符串处理：字符串查找、扫描、转换等
- I/O操作：与外设接口进行数据/状态/命令信息的交换
- 程序流控制：条件转移、无条件转移、转子、返回等
- 系统控制：启动、停止、陷阱指令（自愿访管）、空操作等

◆ 操作数类型（以Pentium处理器数据类型为例）

- 序数或指针：8位、16位、32位无符号整数表示
- 整数：16位、32位、64位三种补码表示的整数
- 实数：IEEE754浮点数格式
- 十进制数：18位十进制数，用80个二进位表示
- 字符串：字节为单位的字符序列，一般用ASCII码表示

◆ 操作数宽度：有多种，如：字节、16位、32位、64位等

本章总结2

作业：习题2(4)、2(9)、3、6、7、8、10、11、13、15、17。11.29晚上24点截止

◆ 寻址方式

- 立即：地址码直接给出操作数本身
- 直接：地址码给出操作数所在的内存单元地址
- 间接：地址码给出操作数所在的内存单元地址所在的内存单元地址
- 寄存器：地址码给出操作数所在的寄存器编号
- 寄存器间接：地址码给出操作数所在单元的地址所在的寄存器编号
- 栈：操作数约定在栈中，总是从栈顶取数或存数
- 偏移寻址：用基地址+形式地址得到操作数所在的内存单元地址

◆ 指令系统：决定了处理器的设计

– 按地址码指定风格来分

累加器型：一个操作数和结果都隐含在累加器中

堆栈型：操作数和结果都隐含在堆栈中

通用寄存器型：操作数明显地指定在哪个通用寄存器中

装入/存储型：运算类指令的操作数只能在寄存器中，只有装入(Load)指令和存储(Store)指令才能访问内存

– 按指令系统的复杂度来分

CISC：复杂指令系统计算机

RISC：精简指令系统计算机