北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

RISC-V指令与仿真

主讲人：陈康冰

计算机学院

德以明理 学以精工

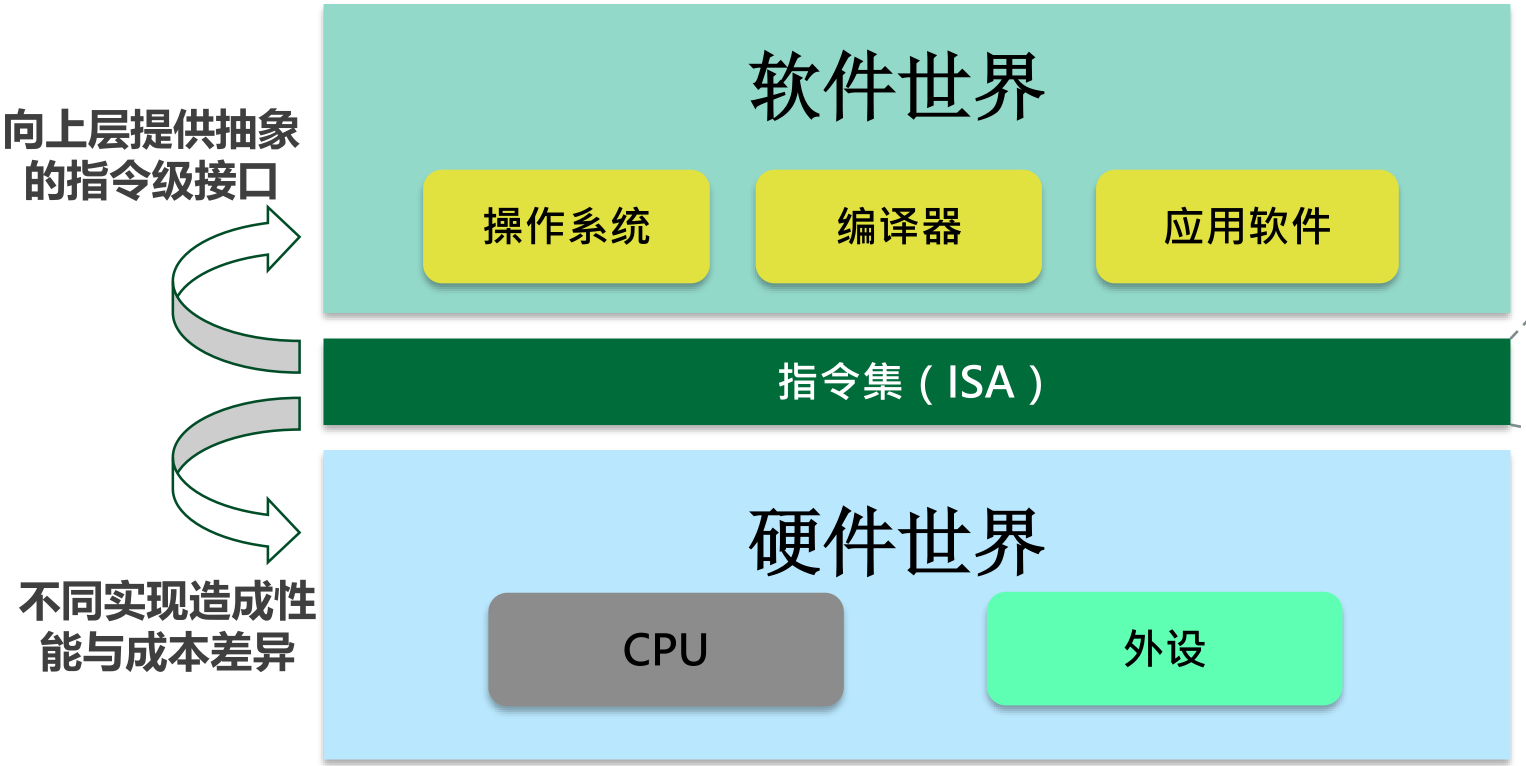
实验二、单周期的处理器设计实验

- 1、学习RISC-V 32I指令，学习使用RISC-V仿真器。
- RARS下载地址：https://github.com/TheThirdOne/rars，
- RARS需要配置Java环境，jdk下载地址：https://www.oracle.com/java/technologies/javase-jdk11-downloads.html
- 2、设计并实现单周期处理器，支持RISCV指令子集：LW，SW，ADD，SUB， ORI，BEQ，LUI，JAL（设计方法请参考王党辉等译，[美] David A. Patterson, John L. Hennessy计算机组成与设计-硬件/软件接口（原书第五版），北京：机械工业出版社，2016年)
- 3、测试程序：完成2后在该cpu上实现对5个整数的排序，仿真测试结果与RARS中运行结果对比正确。
- 4、乐学提交实验报告、工程源代码和测试程序。

目录 | CONTENTS

1 RISC V简要介绍

ISA-从中间层设计计算机系统



CISC

- 复杂指令集
- 指令庞大、结构复杂、功耗高
- x86系列（Intel/AMD）

RISC

- 精简指令集
- 指令集小、结构简单、功耗低
- MIPS、ARM、RISC-V

从CISC到RISC：随着CISC指令集的不断丰富，越来越多的指令被加入到架构之中，而典型程序的运算过程中所使用到的80%指令只占所有指令的20%。CISC在硬件设计、时间与成本上的弊端让RISC指令集架构逐渐成为现代指令集架构设计和选择的主流。

RISCV发展历程

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

1981

RISC I发表在 计算机协会 （ACM）国际计算机体系结构研讨会（ISCA）上

1983-1988

RISC II、RISC III、RISC IV等数代RISC指令集陆续被发明和提出

2010

David Patterson教授与Krste Asanovic教授研究团队开始建立RISCV

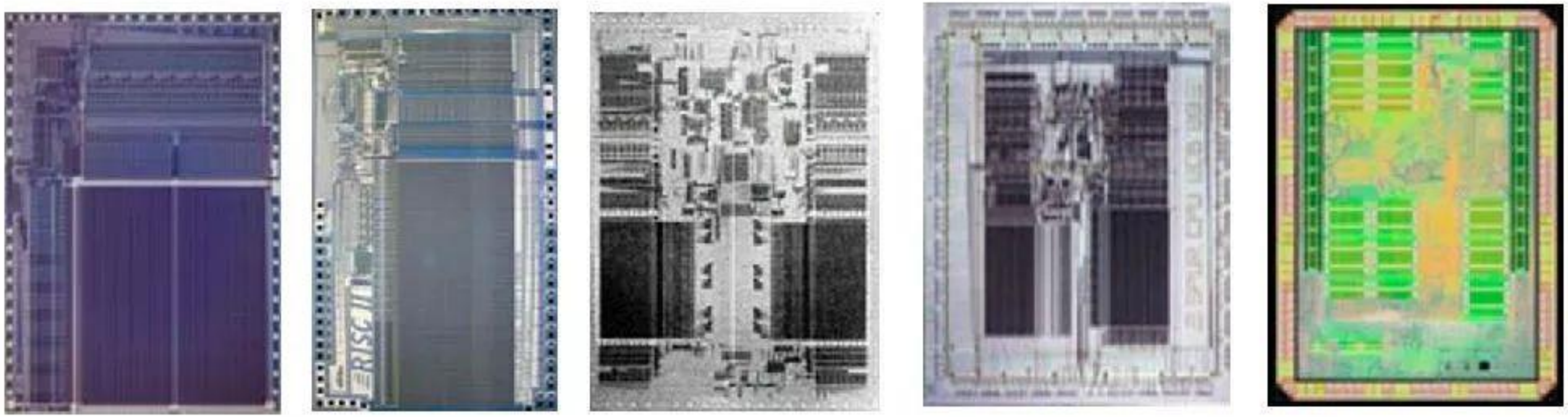
2011

RISCV初版标准发布

2015

伯克利研究团队成立了SiFive初创公司，加速RISC-V的商业化进程

RISC V五代



RISC-I 1981 | RISC-II 1983 | RISC-III (SOAR) 1984 | RISC-IV (SPUR) 1988 | RISC-V 2013

数据格式与地址对齐

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

数据格式定义

RISCV定义了下面几种格式的数据

- Bit (b)
- Byte (8 bits, B)
- Halfword (16 bits, H)
- Word (32 bits, W)
- Doubleword (64 bits, D)

地址对齐约束

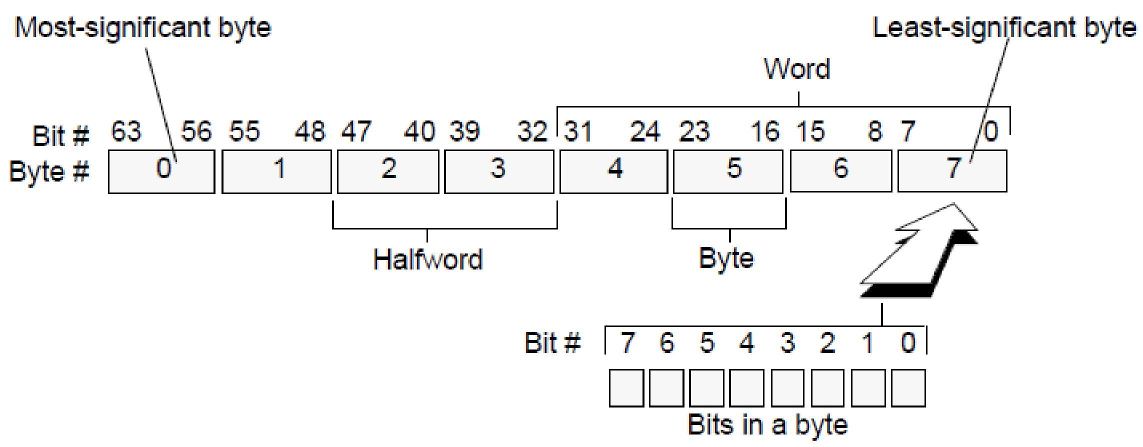
当存储的数据大于一个字节时需要在内存中对齐特定的边界

- 半字必须存储在偶字节地址上(0,2,4...)
- 字必须存储在能被4整除的地址上(0,4,8...)
- 双字必须存储在能被8整除的地址上(0,8...)

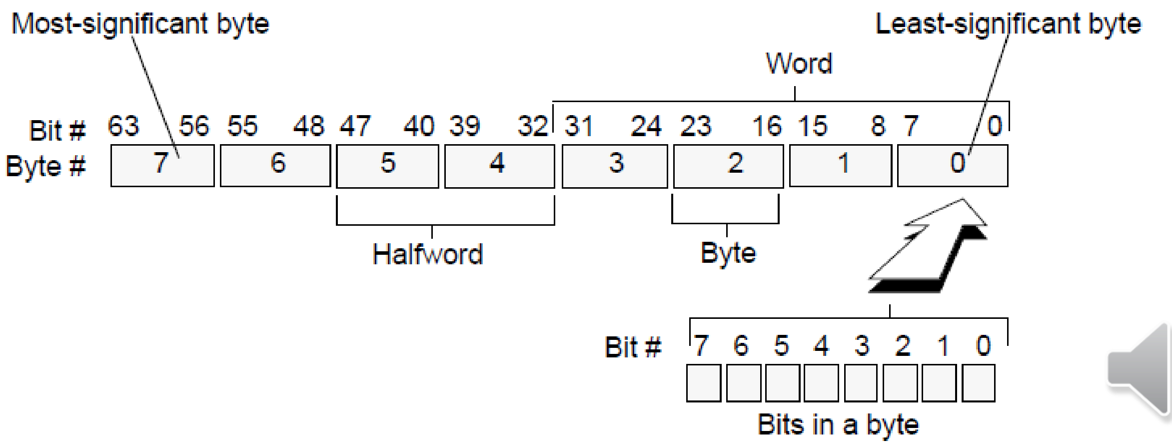
字节顺序选择

字节顺序是指占内存多于一个字节类型的数据在内存中的存放顺序。

大端(Big-Endian)



小端(Little-Endian)



实验统一采用小端字节序

举个例子，依次存放两个word数据在按字节编址的内存中：0x44434241、0x4D3C2B1A

大端(Big-Endian)

	…	高地址
	0x1A	
	0x2B	
	0x3C	
	0x4D	
⋮	0x41	
	0x42	
0x00000001	0x43	
0x00000000	0x44	低地址

小端(Little-Endian)

	…	高地址
	0x4D	
	0x3C	
	0x2B	
	0x1A	
⋮	0x44	
	0x43	
0x00000001	0x42	
0x00000000	0x41	低地址

2 RV32I指令集与寄存器

六种指令格式

- I型指令（I-type）

31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode

- R型指令（R-type）

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode

- S型指令（S-type）

31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode

- B型指令（B-type）

31	30 25	24 20	19 15	14 12	11 8	7	6 0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode

- U型指令（U-type）

31 12	11 7	6 0
imm[31:12]	rd	opcode

- J型指令（J-type）

31 12	11 7	6 0
imm[20|10:1|11|19:12]	rd	opcode

指令中各字段的含义

31	30–25	24–21	20	19–15	14–12	11–8	7	6–0		
funct7		rs2		rs1	funct3	rd		opcode	R-type	寄存器-寄存器操作
imm[11:0]				rs1	funct3	rd		opcode	I-type	短立即数和访存load
imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode	S-type	访存store指令
imm[12]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode	B-type	条件跳转指令
imm[31:12]						rd		opcode	U-type	长立即数
imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode	J-type	无条件跳转

字段	具体描述
opcode	7-bit指令操作码
rd	5-bit目的寄存器定位符
rs1	5-bit源寄存器定位符或在内存访问时指定内存地址
rs2	5-bit源寄存器定位符
imm	立即数，用作算术逻辑、访存、分支跳转等操作
funct3	3-bit功能码，对于相同的opcode字段，通过此字段明确具体功能
funct7	7-bit功能码，对于相同的opcode字段和funct3字段，通过此字段进一步明确具体功能

指令转机器码

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
add	ADD	R	0110011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0110011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0110011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0110011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0110011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0110011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0110011	0x5	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0110011	0x5	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2	0x00	rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3	0x00	rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0		rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x4		rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x6		rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x7		rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	imm[5:11]=0x00	rd = rs1 << imm[0:4]	
srli	Shift Right Logical Imm	I	0010011	0x5	imm[5:11]=0x00	rd = rs1 >> imm[0:4]	
srai	Shift Right Arith Imm	I	0010011	0x5	imm[5:11]=0x20	rd = rs1 >> imm[0:4]	msb-extends
slti	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltiu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≥	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1 + imm	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	imm=0x0	Transfer control to OS	
ebreak	Environment Break	I	1110011	0x0	imm=0x1	Transfer control to debugger	

For example: **add x9, x20, x21**

This is an R-type instruction:

funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

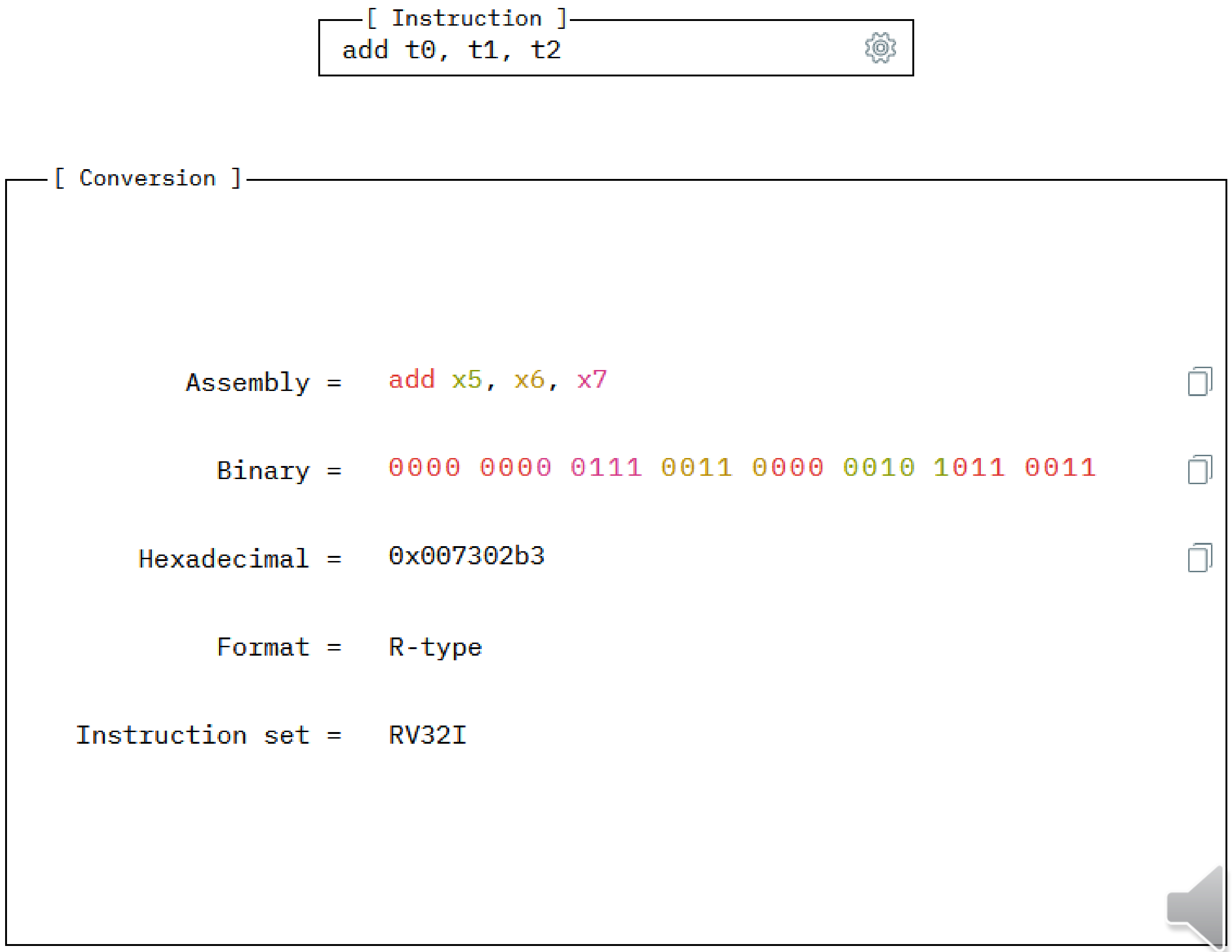
Its binary code is

0000000	10101	10100	000	01001	0110011

The hexadecimal code for this instruction is: 0x015A04B3

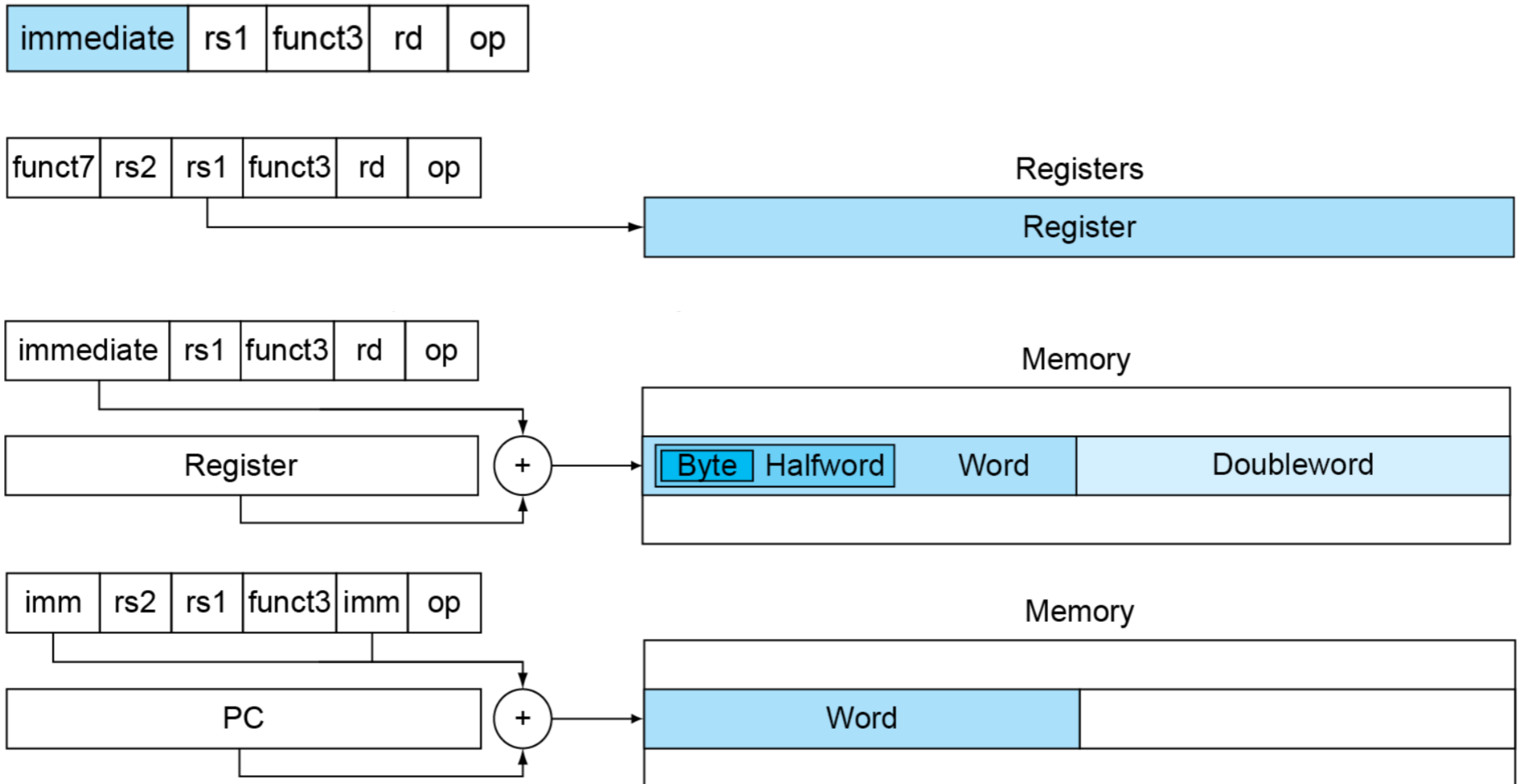
指令转机器码

https://luplab.gitlab.io/rvcodecjs
在线RISC-V指令机器码转换



RISCV寻址方式

1. 立即数寻址
2. 寄存器寻址
3. 基址寻址
4. PC相对寻址



RISCV基础指令概览（包括但不限于）

算术运算指令

add/addi/sub/
slt/slti/sltiu

逻辑运算指令

and/andi/lui
or/ori
xor/xori

位移运算指令

sll/slli
sra/srai
srl/srli

分支跳转指令

beq/bne/bge/bgeu
blt/bltu
jal/jalr

内存访问指令

lb/lbu/sb
lh/lhu/sh
lw/sw

特殊指令

fence/ecall/ebreak
csrrw/csrrs/csrrc

算术运算指令举例

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

ADDI

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[11:0]			rs1	000	rd	00100	11

汇编格式：ADDI rd, rs1, imm
功能描述：将寄存器rs1的值与有符号扩展至32位的立即数imm相加，结果写入rd寄存器中。
指令格式：I-type
寻址方式：立即数寻址、寄存器寻址

SUB

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
01000	00	rs2	rs1	000	rd	01100	11

汇编格式：SUB rd, rs1, rs2
功能描述：将寄存器rs1的值与寄存器rs2的值相减，结果写入rd寄存器中。
指令格式：R-type
寻址方式：寄存器寻址

算术运算指令举例

- ADDI

[Instruction]

addi t0, t1, 100

[Conversion]

Assembly = addi x5, x6, 100

Binary = 0000 0110 0100 0011 0000 0010 1001 0011

Hexadecimal = 0x06430293

Format = I-type

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[11:0]			rs1	000	rd	00100	11

序号	指令格式	具体指令	imm (31-20)	rs1 (19-15)	funct3 (14-12)	rd (11-7)	opcode (6-0)
1	addi rd, rs, imm	addi t0, t1, 100	0000 0110 0100	00110	000	00101	0010011
2							
3							
4							

逻辑运算指令举例

XOR

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	00	rs2	rs1	100	rd	01100	11

汇编格式：XOR rd, rs1, rs2
功能描述：寄存器rs1中的值与寄存器rs2中的值按位逻辑异或，结果写入寄存器rd中。
指令格式：R-type
寻址方式：寄存器寻址

LUI

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
imm[31:12]					rd	01101	11

汇编格式：LUI rt, imm
功能描述：将20位立即数imm写入寄存器rt的高20位，寄存器rt的低12位置0。
指令格式：I-type
寻址方式：立即数寻址

位移运算指令举例

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

SLL

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	00	rs2	rs1	001	rd	01100	11

汇编格式：SLL rd, rs1, rs2
功能描述：由寄存器rs2的低5位指定移位量，对寄存器rs1的值进行逻辑左移，结果写入寄存器rd中。
指令格式：R-type
寻址方式：寄存器寻址

SRAI

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
01000	00	shamt	rs1	101	rd	00100	11

汇编格式：SRAI rd, rs1, shamt
功能描述：由5位短立即数shamt指定移位量，对寄存器rs1的值进行算术右移，结果写入寄存器rd中。
指令格式：I-type
寻址方式：立即数寻址、寄存器寻址

分支跳转指令举例

■ BGE

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[12|10:5]		rs2	rs1	101	offset[4:1|11]	11000	11

汇编格式：BGE rs1, rs2, offset
功能描述：如果寄存器rs1的值大于等于rs2的值则转移，否则顺序执行。转移目标由立即数offset取低12位至低1位（即最低位置0之后的低12位）并进行有符号扩展的值加上该分支指令的PC计算得到。
指令格式：B-type
寻址方式：寄存器寻址、PC相对寻址

跳转范围：
±4KB

■ JAL

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[20|10:1|11			19:12]		rd	11011	11

汇编格式：JAL target
功能描述：无条件跳转。跳转目标由立即数offset取低20位至低1位（即最低位置0之后的低20位）并进行有符号扩展的值加上该分支指令的PC计算得到，并将返回地址（即当前跳转指令的下一条指令对应的PC）保存至寄存器rd，以供之后返回。
指令格式：J-type
寻址方式：PC相对寻址

跳转空间：
±1MB

内存访问指令举例

SW

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[11:5]		rs2	rs1	010	offset[4:0]	01000	11

汇编格式：SW rs2, offset(rs1)
功能描述：将rs1寄存器的值加上符号扩展后的立即数offset得到访存的虚地址，如果地址不是4的整数倍则触发地址未对齐异常，否则据此虚地址将rs2寄存器存入存储器中。
指令格式：S-type
寻址方式：基址寻址、寄存器寻址

LBU

31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
offset[11:0]			rs1	100	rd	00000	11

汇编格式：LBU rd, offset(rs1)
功能描述：将rs1寄存器的值加上符号扩展后的立即数offset得到访存的虚地址，据此虚地址从存储器中读取1个字节的值并进行0扩展，写入到rd寄存器中。
指令格式：I-type
寻址方式：基址寻址、寄存器寻址

一些特殊指令与RISCV寄存器

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

- FENCE

  保证内存一致性。

- ECALL

  触发系统调用。

- EBREAK

  触发断点例外。

- CSRR**

  控制状态寄存器访问指令。

- GPR通用寄存器

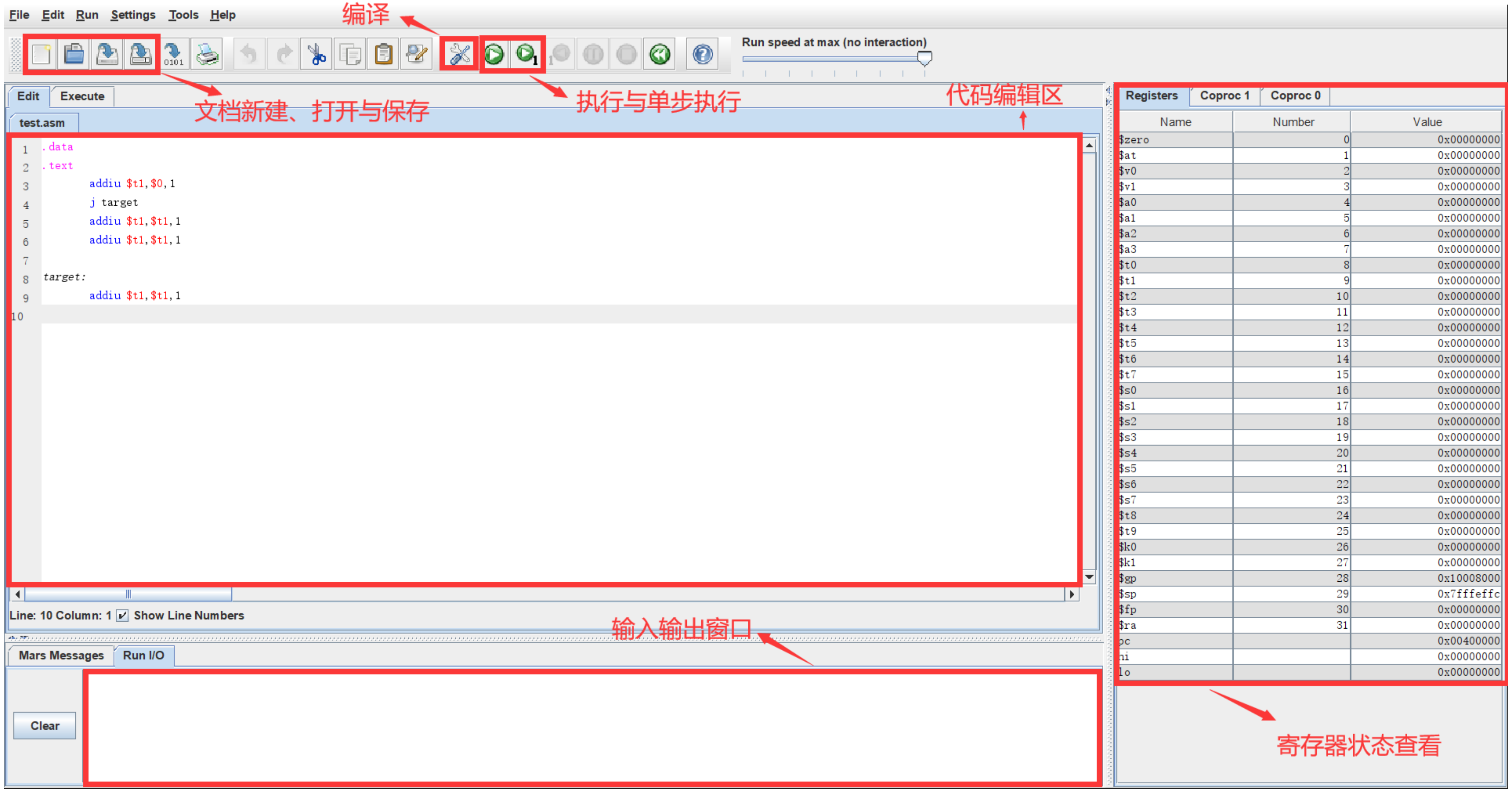
寄存器	ABI名字	描述	保存者
x0	zero（零）	硬件连线0	—
x1	ra	返回地址	调用者
x2	sp	栈指针	被调用者
x3	gp	全局指针	—
x4	tp	线程指针	—
x5-7	t0-2	临时变量	调用者
x8	s0/fp	保存的寄存器/帧指针	被调用者
x9	s1	保存的寄存器	被调用者
x10-11	a0-1	函数参数/返回值	调用者
x12-17	a2-7	函数参数	调用者
x18-27	s2-11	保存的寄存器	被调用者
x28-31	t3-6	临时变量	调用者
f0-7	ft0-7	FP临时变量	调用者
f8-9	fs0-1	FP保存的寄存器	被调用者
f10-11	fa0-1	FP参数/返回值	调用者
f12-17	fa2-7	FP参数	调用者
f18-27	fs2-11	FP保存的寄存器	被调用者
f28-31	ft8-11	FP临时变量	调用者

- PC程序计数器

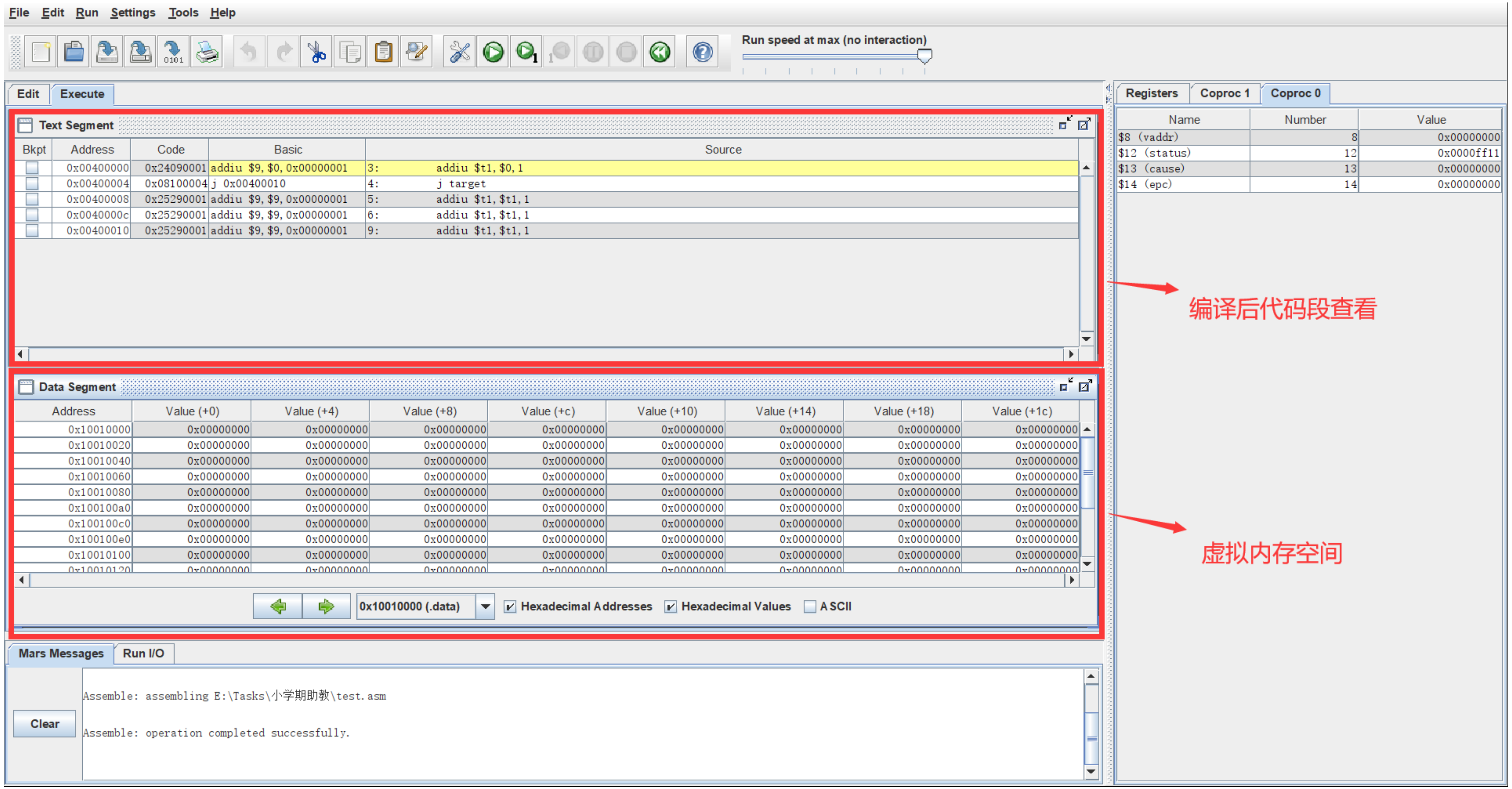
  32位寄存器

3 RARS使用

RISCV汇编模拟器RARS



RISCV汇编模拟器RARS

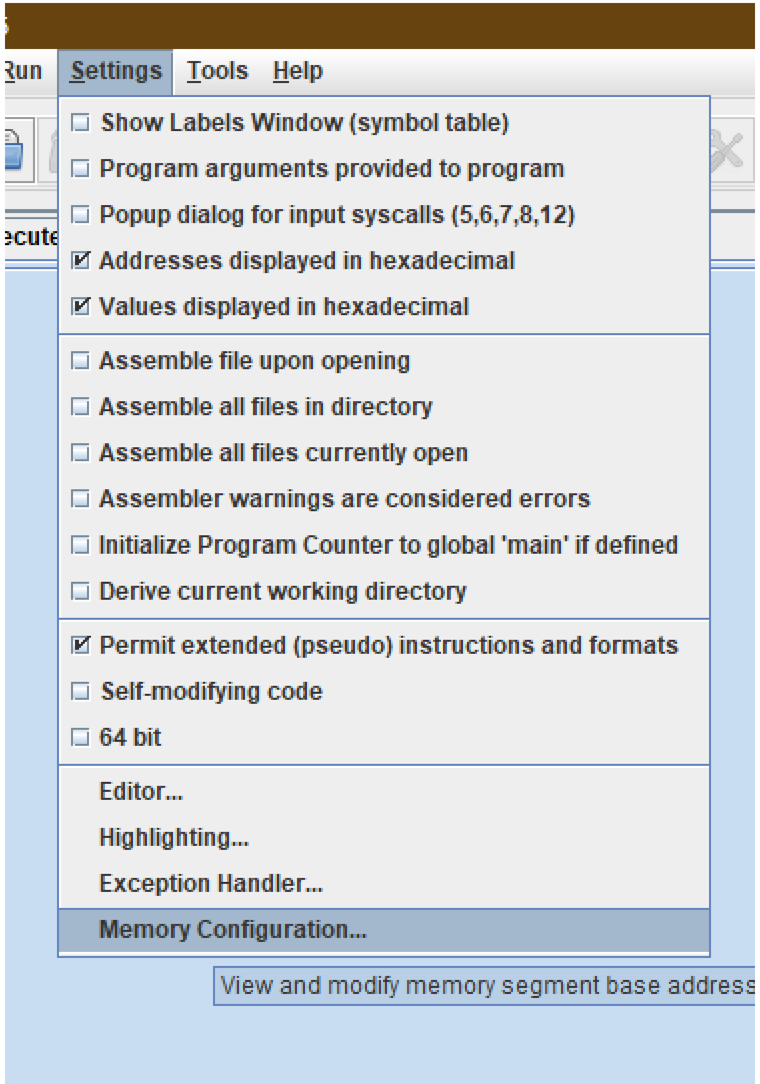


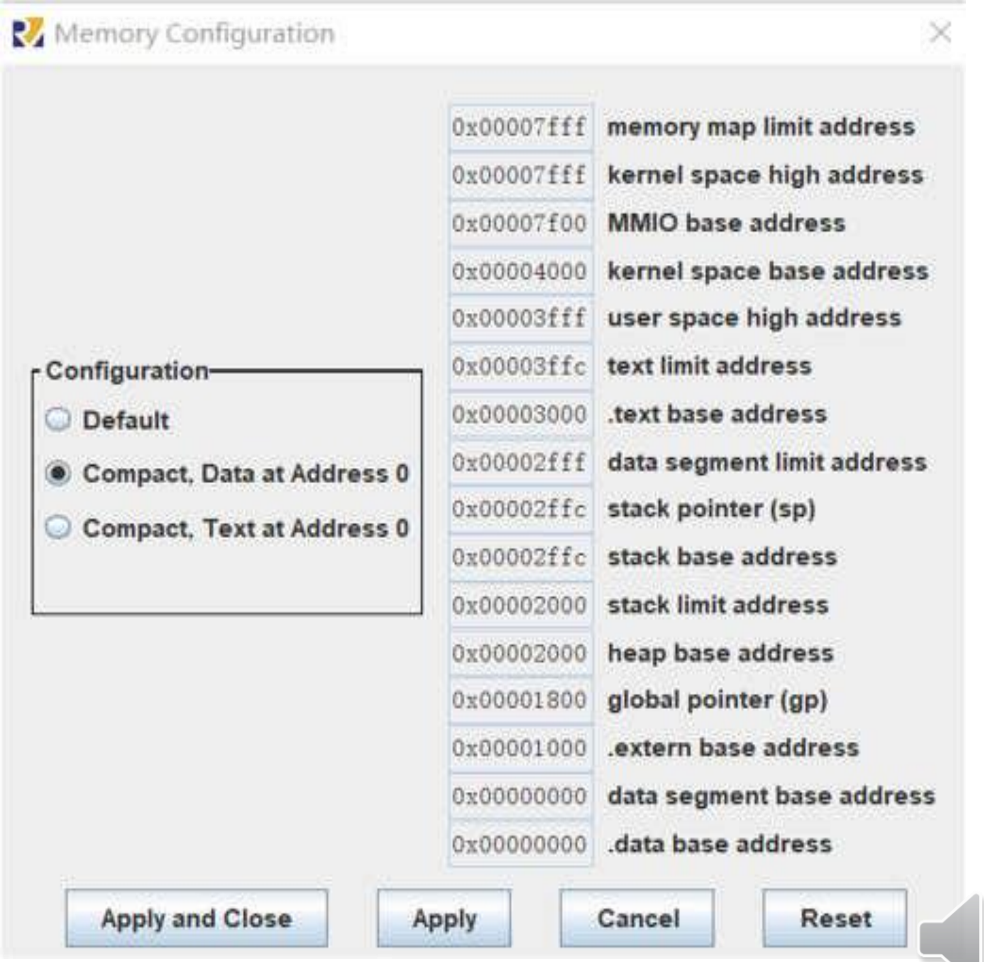
编译后代码段查看

虚拟内存空间

注：RARS使用需要有java环境

RISCV汇编模拟器RARS





RISCV汇编模拟器RARS

Edit | Execute

riscv1.asm

```
.data
.text
        addi t1, zero, 1
        jal target
        addi t1, t1, 1
        addi t1, t1, 1

target:
        addi t1, t1, 1

```

Edit | Execute

Text Segment

Bkpt	Address	Code	Basic	
☐	0x00400000	0x00100313	addi x6, x0, 1	3: addi t1, zero, 1
☐	0x00400004	0x00c000ef	jal x1, 0x0000000c	4: jal target
☐	0x00400008	0x00130313	addi x6, x6, 1	5: addi t1, t1, 1
☐	0x0040000c	0x00130313	addi x6, x6, 1	6: addi t1, t1, 1
☐	0x00400010	0x00130313	addi x6, x6, 1	9: addi t1, t1, 1

RISCV系统调用

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

输入输出

syscall调用系统服务

Step1. 服务编码存入a7。

Step2. 需要传递的参数存入a0, a1, a2, a3, fa0, ... 中（如果有）。

Step3. ecall

Step4. 查看返回结果（如果有）。

服务	编码	参数传递	结果返回
打印整数	1	a0 = 要打印的整数值	
打印单精度浮点数	2	fa0 = 要打印的浮点数	
打印双精度浮点数	3	fa0 = 要打印的浮点数	
打印字符串	4	a0 = 字符串首地址	
读取整数	5		a0 携回读入的整数
读取单精度浮点数	6		fa0 携回读入的浮点数
读取双精度浮点数	7		fa0 携回读入的浮点数
读入字符串	8	a0 = 输入缓冲区地址 a1 = 最大读入字符个数	指定区域携回字符串
结束进程	10		
打印字符	11	a0 = 要打印的字符	
读取字符	12		a0 携回读入的字符

Hello World!

Edit　Execute

riscv1.asm

```
.data
        message: .ascii "Hello, World!\n"

.text
main:
        li a7, 4                # 4号系统调用，打印字符串
        la a0, message          # 保存字符串地址
        ecall                   # 调用系统调用

exit:
        li a7, 10               # 10号系统调用，退出程序
        ecall
```

编译

Address	Code	Basic	Sou
0x00400000	0x00400893	addi x17, x0, 4	6: li a7, 4
0x00400004	0x0fc10517	auipc x10, 0x0000fc10	7: la a0, message
0x00400008	0xffc50513	addi x10, x10, 0xfffffffc	
0x0040000c	0x00000073	ecall	8: ecall
0x00400010	0x00a00893	addi x17, x0, 10	11: li a7, 10
0x00400014	0x00000073	ecall	12: ecall

(+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+1
0x6c6c6548	0x57202c6f	0x646c726f	0x00000a21	0x00000000	0x00000000	0
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0

0x10010000 (.data)　☑ Hexadecimal Addresses　☑ Hexadecimal Values　☐ ASCII

执行

Messages　Run I/O

```
Hello, World!

-- program is finished running (0) --
```

封装字符串打印

printf ?

对特定过程进行封装可以进行代码重用，缩小程序规模，减少指令段的内存占用。

Edit Execute

riscv1.asm

```
.data
        message: .ascii "Hello, World!\n"

.text
main:
        li a7, 4                # 4号系统调用，打印字符串
        la a0, message          # 保存字符串地址
        ecall                   # 调用系统调用

exit:
        li a7, 10               # 10号系统调用，退出程序
        ecall

```

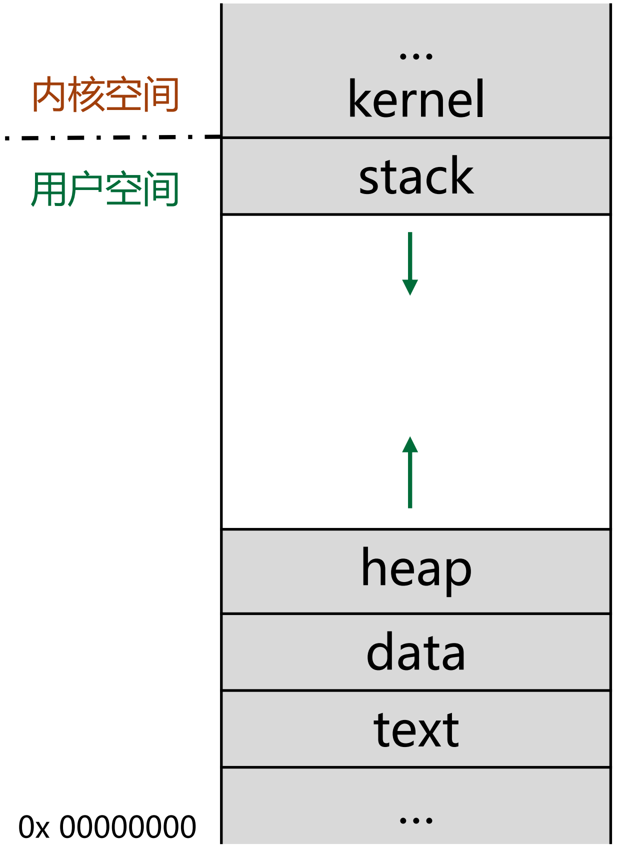
过程封装

Edit Execute

riscv1.asm

```
.data
        message: .ascii "Hello, World!\n"

.text
main:
        la a0, message          # 保存字符串地址
        jal printString         # 调用函数

exit:
        li a7, 10               # 10号系统调用，退出程序
        ecall

printString:
        li a7, 4                # 4号系统调用，打印字符串
        ecall

```

过程调用



调用与返回

调用方调用某个过程： jal label
过程指令结束后返回： jalr t1

传参与接受返回值

调用方调用过程之前传参到： a0|a1|a2|...
过程返回值保存到： a0|a1

保存现场入栈

调用方需要保存的数据存到： s0-s11

```
push:   sub sp, sp, 4
        sw s0, 0(sp)
pop:    lw s0, 0(sp)
        add sp, sp, 4
```

涉及到嵌套过程调用时还需保存： ra, sp, gp, tp

排序例程

```
.data
list:
    .word 4, 2, 8, 5, 7, 1

.text
bubsort:
    la a0, list     # a0 = *list
    li a1, 6        # a1 = size

loop1_start:
    # do loop
    li t0, 0        # swapped = false
    li t1, 1        # i = 1

loop2_start:
    # for loop
    bge t1, a1, loop1_end   # break if i >= size
    slli t3, t1, 2          # scale i by 4 (for word)
    add t3, a0, t3          # new scaled memory address
    lw  t4, -4(t3)          # load list[i-1] into t4
    lw  t5, 0(t3)           # load list[i] into t5
    ble t4, t5, loop2_end   # if list[i-1] < list[i], it's in position
    # if we get here, we need to swap
    li  t0, 1               # swapped = true
    sw  t4, 0(t3)           # list[i] = list[i-1]
    sw  t5, -4(t3)          # list[i-1] = list[i]

loop2_end:
    # bottom of for loop body
    addi t1, t1, 1          # i++
    jal loop2_start         # loop again

loop1_end:
    # bottom of do loop body
    bnez t0, loop1_start    # loop if swapped = true

stop:
    jal stop                # stop

```

排序例程

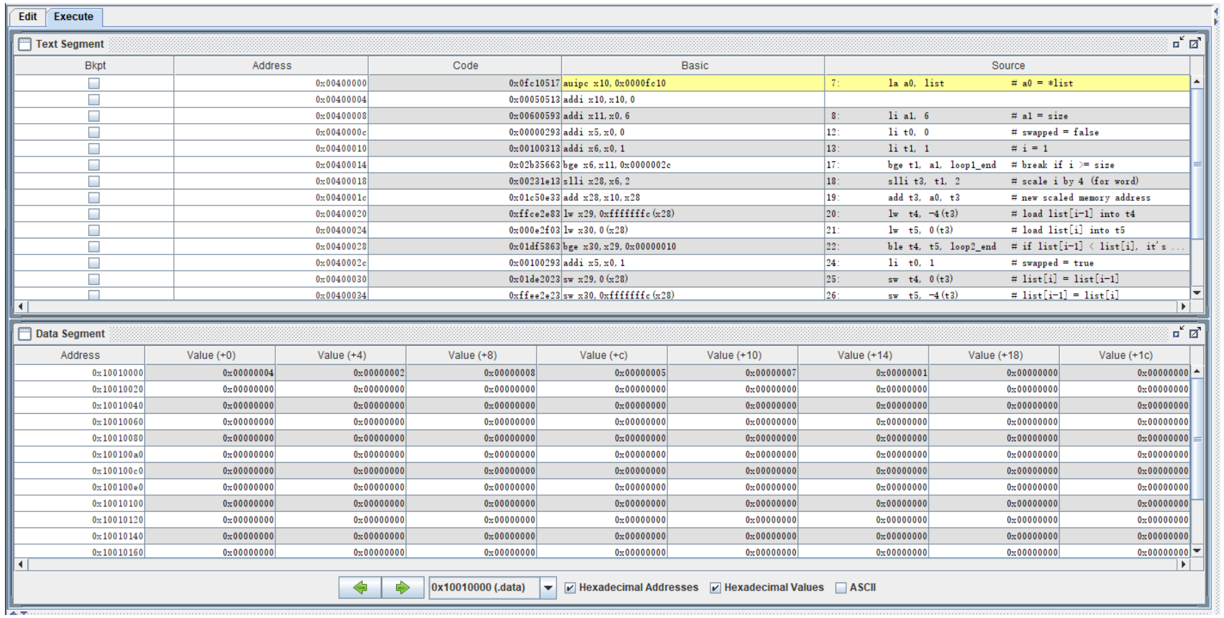
Edit Execute

sort.asm

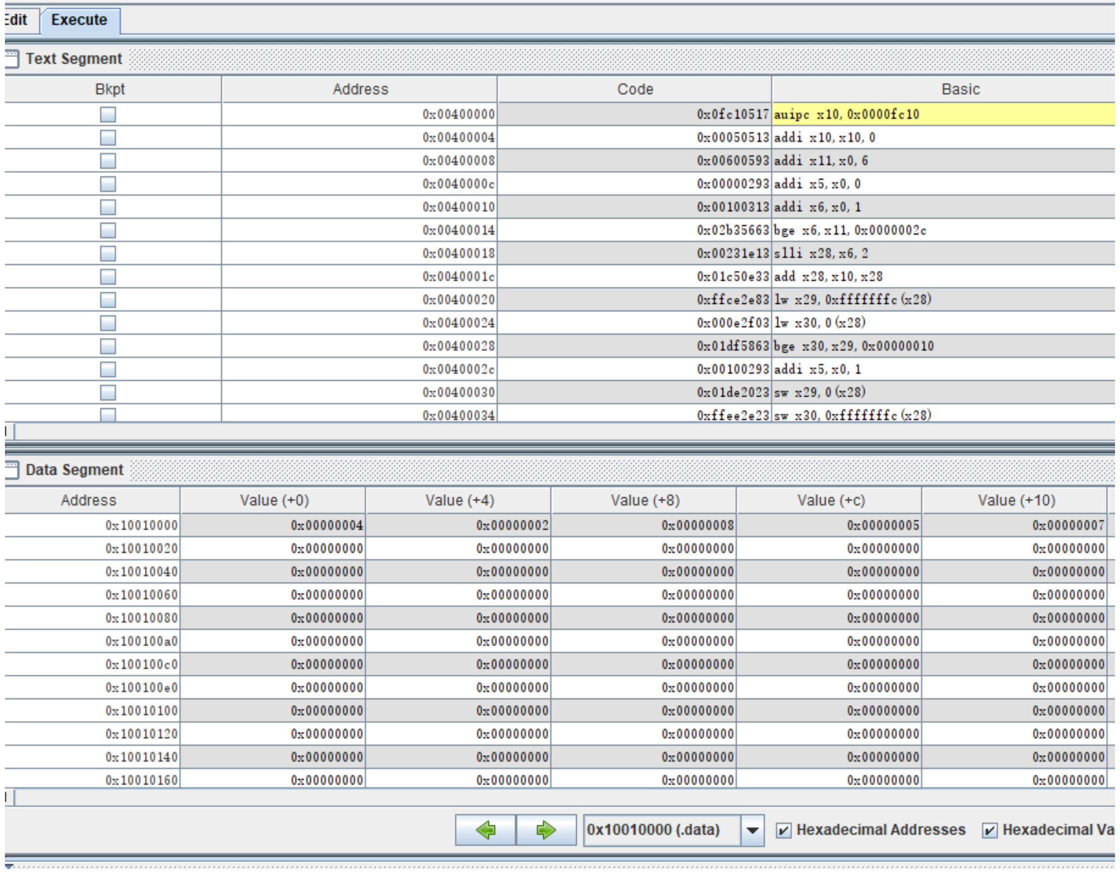
```
.data
list:
        .word 4, 2, 8, 5, 7, 1

.text
bubsort:
        la a0, list             # a0 = *list
        li a1, 6                # a1 = size

loop1_start:
        # do loop
        li t0, 0                # swapped = false
        li t1, 1                # i = 1

loop2_start:
        # for loop
        bge t1, a1, loop1_end   # break if i >= size
        slli t3, t1, 2          # scale i by 4 (for word)
        add t3, a0, t3          # new scaled memory address
        lw  t4, -4(t3)          # load list[i-1] into t4
        lw  t5, 0(t3)           # load list[i] into t5
        ble t4, t5, loop2_end   # if list[i-1] < list[i], it's in position
        # if we get here, we need to swap
        li  t0, 1               # swapped = true
        sw  t4, 0(t3)           # list[i] = list[i-1]
        sw  t5, -4(t3)          # list[i-1] = list[i]
```

Line: 24 Column: 14 Show Line Numbers

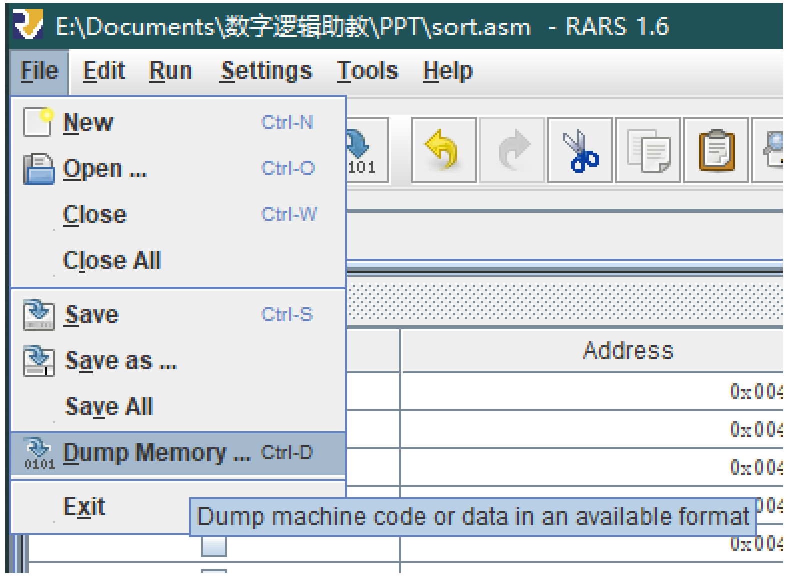
编译

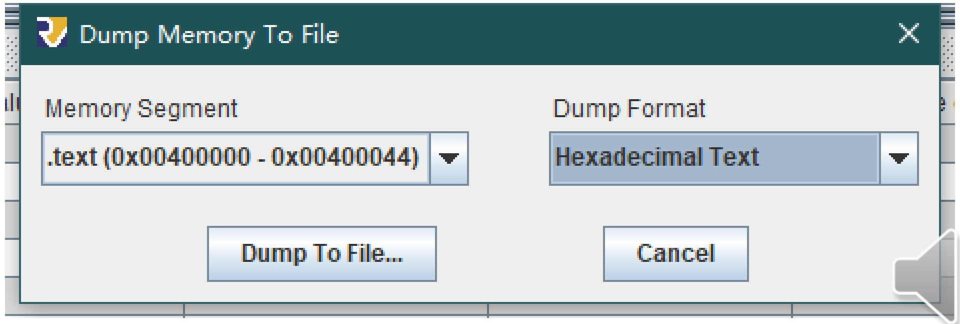


排序例程


Edit
Execute
Text Segment
Bkpt
Address
Code
Basic
0x00400000 0x0fc10517 auipc x10, 0x0000fc10
0x00400004 0x00050513 addi x10, x10, 0
0x00400008 0x00600593 addi x11, x0, 6
0x0040000c 0x00000293 addi x5, x0, 0
0x00400010 0x00100313 addi x6, x0, 1
0x00400014 0x02b35663 bge x6, x11, 0x0000002c
0x00400018 0x00231e13 slli x28, x6, 2
0x0040001c 0x01c50e33 add x28, x10, x28
0x00400020 0xffce2e83 lw x29, 0xfffffffc(x28)
0x00400024 0x000e2f03 lw x30, 0(x28)
0x00400028 0x01df5863 bge x30, x29, 0x00000010
0x0040002c 0x00100293 addi x5, x0, 1
0x00400030 0x01de2023 sw x29, 0(x28)
0x00400034 0xffee2e23 sw x30, 0xfffffffc(x28)
Data Segment
Address
Value (+0)
Value (+4)
Value (+8)
Value (+c)
Value (+10)
0x10010000 0x00000004 0x00000002 0x00000008 0x00000005 0x00000007
0x10010020 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010040 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010060 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010080 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x100100a0 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x100100c0 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x100100e0 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010100 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010120 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010140 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010160 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010000 (.data)
Hexadecimal Addresses
Hexadecimal Va

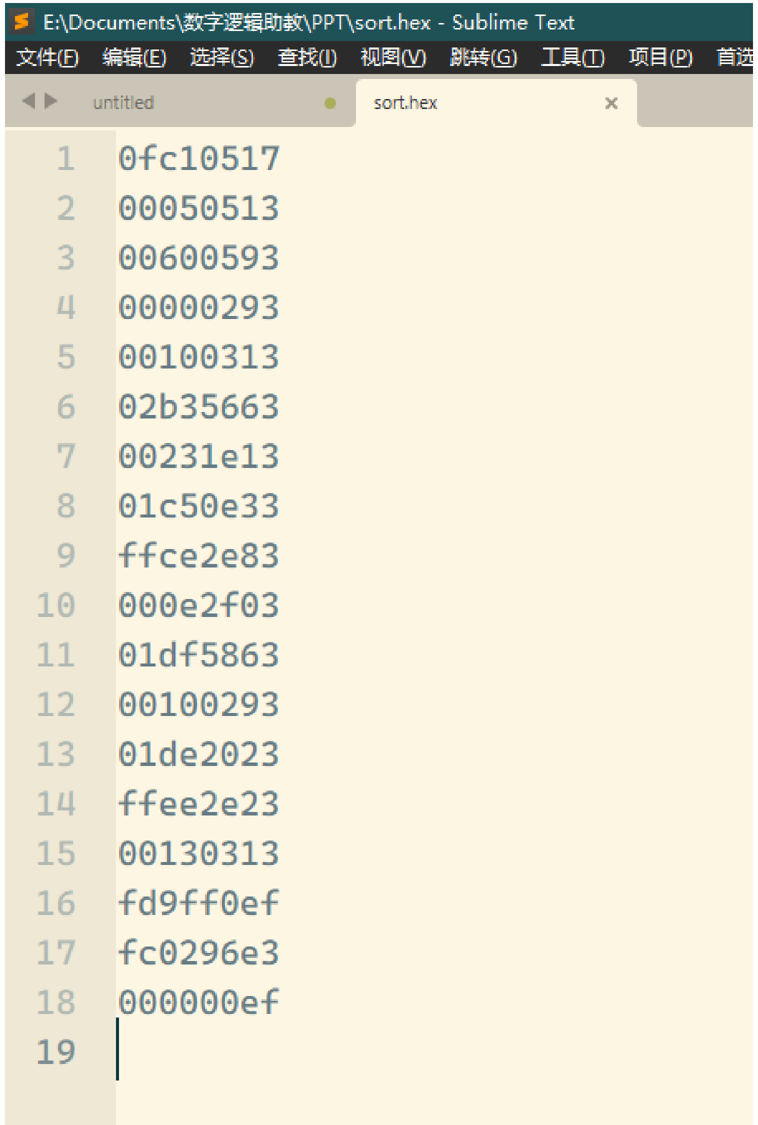
导出


E:\Documents\数字逻辑助教\PPT\sort.asm - RARS 1.6
File
Edit
Run
Settings
Tools
Help
New Ctrl-N
Open ... Ctrl-O
Close Ctrl-W
Close All
Save Ctrl-S
Save as ...
Save All
Dump Memory ... Ctrl-D
Exit
Dump machine code or data in an available format
Address


Dump Memory To File
Memory Segment
.text (0x00400000 - 0x00400044)
Dump Format
Hexadecimal Text
Dump To File...
Cancel

排序例程

导出



在CPU上运行……

$readmemh()

或

IP核 .coe文件导入

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

感谢各位

主讲人：陈康冰

计算机学院

德以明理　学以精工