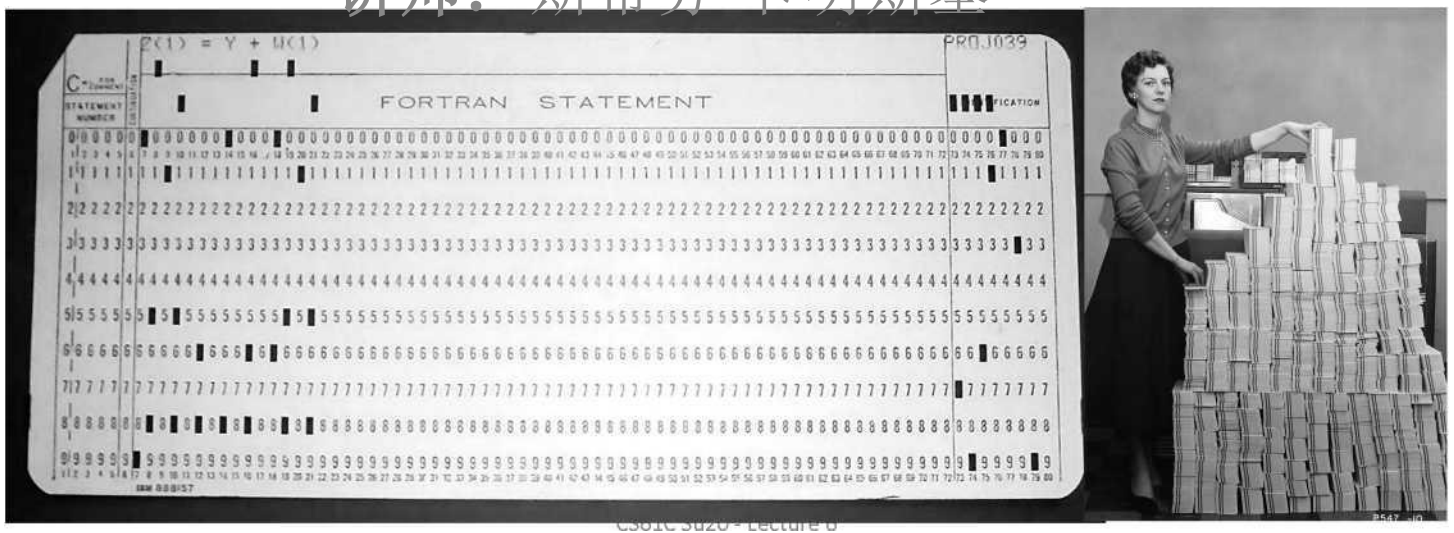
Great Ideas in Computer Architecture

机器语言简介：RISC-V

讲师：斯蒂芬·卡明斯基



上次讲座回顾

- C 内存布局

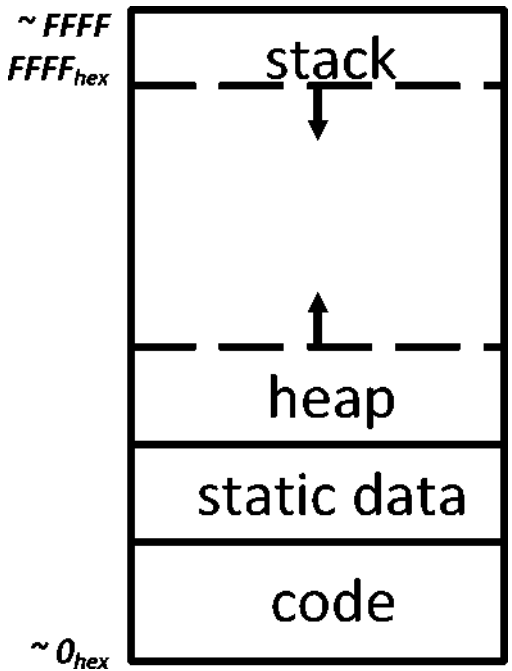
—Stack：局部变量

—静态数据：全局变量

—代码：机器指令

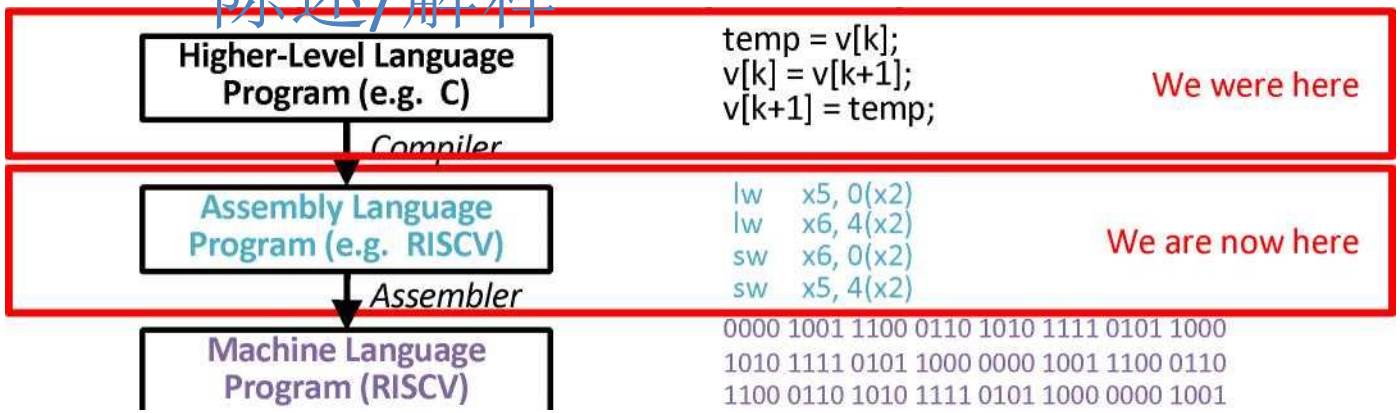
— 堆：使用 malloc 和 free 的动态存储

- 必须小心使用



好主意#1：级别

陈述/解释



机器

解释

硬件体系结构描述
（如方框图）

体系结构

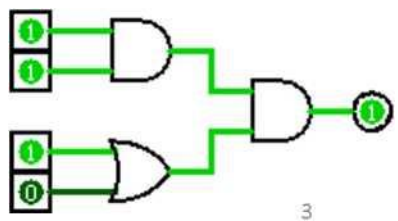
实施

逻辑电路描述
（电路原理图）

寄存器文件

1 1

ALU



组件（也称为：汇编语言（ASM）

- 一种低级程序设计语言，其程序指令与特定体系结构的操作相匹配。
- 将一个程序拆分为许多小指令，每个小指令完成程序的一个单独部分

C 程序汇编程序

```
a  =(b+c)-(d+e);  add  t1, s3, s4
                  add t2, s1, s2
                  sub s0, t2, t1
```

有很多汇编语言

- 一种低级程序设计语言，其程序指令与特定体系结构的操作相匹配。
- 每个体系结构将具有它所支持的一组不同的操作（尽管有许多相似之处）
- 程序集不可移植到其他体系结构（如 C）

主流指令集体系结构

x86

英特尔、AMD 设计师

位 16 位、32 位和 64 位

1978 年（16 位）、1985 年（32 位）、2003 年推出

（64 位）

设计 CISC

类型寄存器-存储器

编码变量（1 到 15 字节）

字节序小

Macbook 和 PC（酷睿 i3、i5、i7、M）x86 指令

ARM 架构

设计师 ARM Holdings

位 32 位、64 位

1985 年引进;31 年前

设计 RISC

类型寄存器-寄存器

编码 AArch64/A64 和 AArch32/A32 使用 32 位指令，T32（Thumb-2）使用 16 位和 32 位混合指令。ARMv7 用户空间兼容性!1!

Endianness Bi（默认值为小）

智能手机类设备（iPhone、Android）、Raspberry Pi、嵌入式系统 ARM 指令集

RISC V

RISC-V

加州大学伯克利分校设计师

位 32,64,128

2010 年推出

版本 2.2

设计 RISC

类型加载-存储

编码变量

分支比较和分支

字节序小

多功能和开源相对较新，专为云计算而设计，嵌入式系统，学术用途 RISCV 指令集 6

哪些说明应组件包括?

有一些明显有用的说明：

- 加、减和移位
- 读写存储器

但是：

- 如果这两个值相等，则只运行下一条指令
- 同时执行四个成对乘法
- 将两个 ascii 数字相加（‘2’ + ‘3’ = 5）

复杂/精简指令集计算

- 早期趋势：增加越来越多的说明进行精细操作：
  复杂指令集计算（CISC）
  — 难学难懂的语言
  — 编译器的工作量更少
  — 复杂的硬件运行更慢
- 后来，相反的哲学开始占主导地位：*精简指令集计算（RISC）*
  —更简单（更小）的指令集更易于构建快速硬件
  —让软件通过合成简单的操作来完成复杂的操作

RISC 主宰现代计算

这个想法有多重要?

- RISC 架构主导计算
  — ARM 是 RISC,所有智能手机都使用 ARM
- 旧的 CISC 架构（x86）类似于 RISC
  在这些日子里
- 2017 年 Patterson 和图灵奖

轩尼诗

我们将在课堂上使用 RISC-V

- 加州大学伯克利分校第五代 RISC 设计——Krste Asanovic 教授和 Adept 实验室
- 开源指令集规范
- 在工业界和学术界迅速发展

  适用于所有计算级别

  — 嵌入式微控制器到超级计算机

  — 32 位、64 位和 128 位变体

  — 专为学术用途而设计

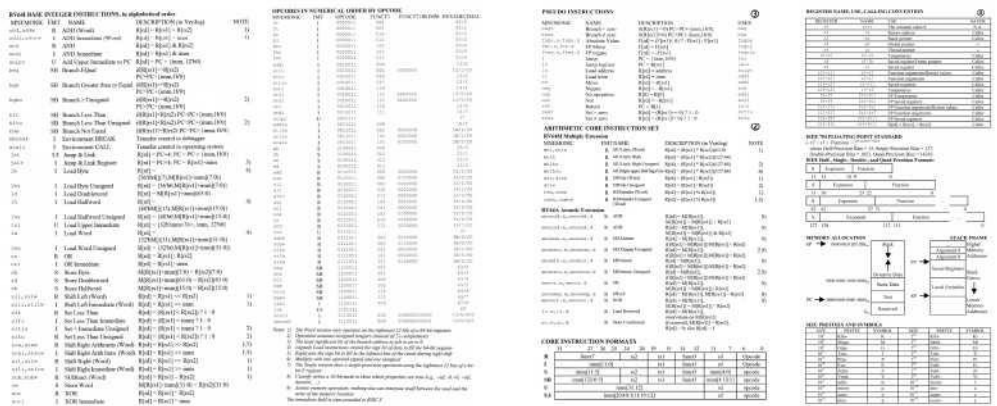
RISC-V 资源

全 RISC-V 架构

http://digitalassets.lib.berkeley.edu/techreports/ucb/text/EECS-2016-1.pdf

CS61C 所需的一切

https://cs61c.org/resources/pdf?file=riscvcard.pdf（“绿卡”）

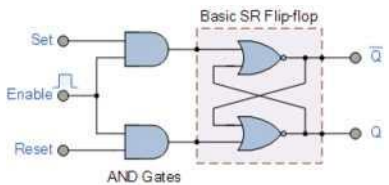


RISCV 议程

- 背景
- **寄存器**
- 装配代码
- 基本算术指令
- 即时指示
- 数据传输说明
- 控制流程说明
- 换档说明
- 其他有用说明
- 总结

硬件将寄存器用于变量

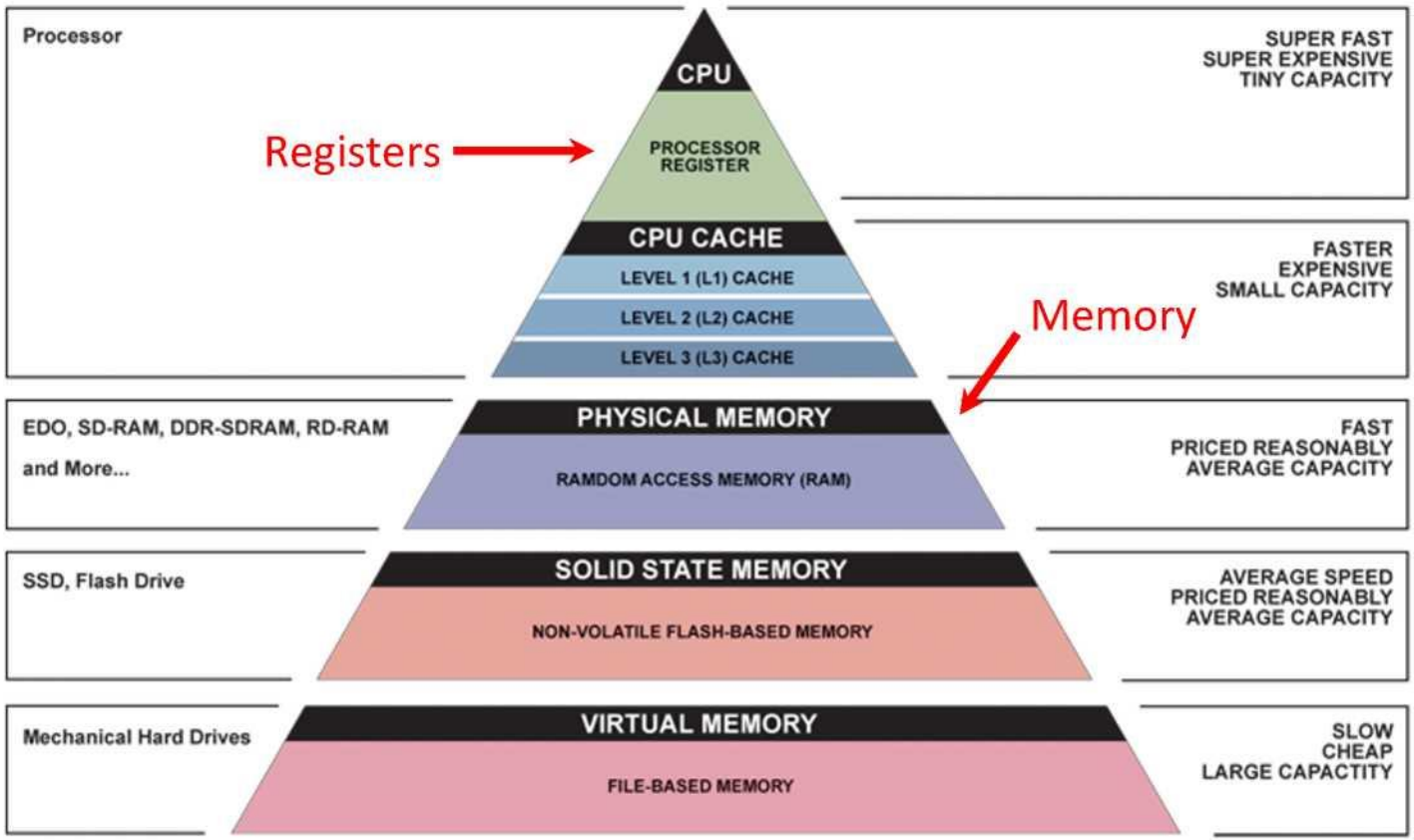
- 与 C 不同，assembly 没有您所知道的变量
- 相反，程序集使用寄存器来存储值
- 寄存器包括：
  - — 固定大小的小型内存（在我们的系统中为 32 位）— 可读取或写入
  - — 数量有限（系统上有 32 个寄存器）
  - — 非常快速、低功耗访问



寄存器与存储器

- 如果变量多于寄存器怎么办?
  — 保留最常用的寄存器并移动
  余下的记忆（称为溢出记忆）
- 为什么不是所有变量都在内存中?
  — 越小越快：寄存器速度加快 100-500 倍
  — 内存层次结构
  - 寄存器：32 个寄存器 * 32 位 = 128 字节
  - RAM：4-32 GB

- SSD：100-1000 GB


Processor
CPU
PROCESSOR REGISTER
Registers
SUPER FAST
SUPER EXPENSIVE
TINY CAPACITY
CPU CACHE
LEVEL 1 (L1) CACHE
LEVEL 2 (L2) CACHE
LEVEL 3 (L3) CACHE
FASTER
EXPENSIVE
SMALL CAPACITY
Memory
EDO, SD-RAM, DDR-SDRAM, RD-RAM
and More...
PHYSICAL MEMORY
RAMDOM ACCESS MEMORY (RAM)
FAST
PRICED REASONABLY
AVERAGE CAPACITY
SSD, Flash Drive
SOLID STATE MEMORY
NON-VOLATILE FLASH-BASED MEMORY
AVERAGE SPEED
PRICED REASONABLY
AVERAGE CAPACITY
Mechanical Hard Drives
VIRTUAL MEMORY
FILE-BASED MEMORY
SLOW
CHEAP
LARGE CAPACTITY

好主意 3：局部性/记忆层次性原则

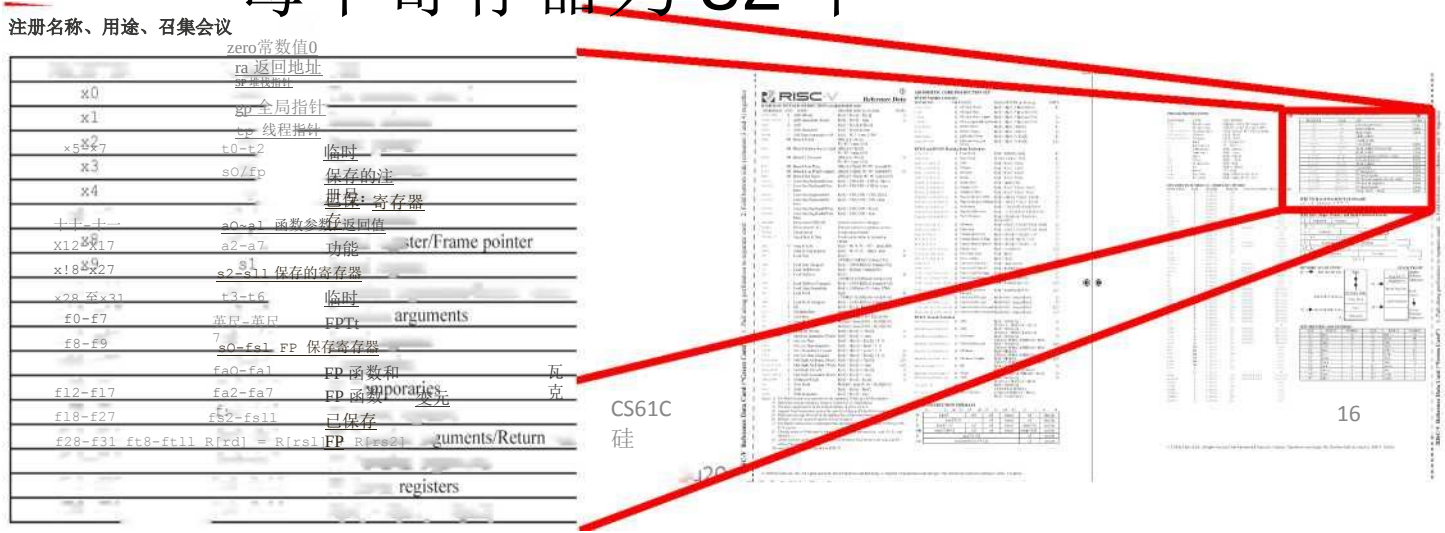
RISCV -- 有多少个寄存器?

- 速度和可用性之间的权衡
  - 更多寄存器 -> 可以容纳更多变量
    - 同时;所有寄存器都较慢。
- RISCV 有 32 个寄存器
  - 每个寄存器为 32 个

注：一个词是固定的-

由处理器的指令集或硬件作为一个单元处理的大小的数据片。正常

一个字被定义为 CPU 寄存器的大小。



RISCV 寄存器

- 以“x”表示的寄存器可通过编号（x0-x31）或名称引用：
  —保存程序员变量的寄存器：

s0-s1 ** x8-x9 »

/Ss2-s11 ~ x18-x27

寄存器‘Name’ <^2— 保存临时变量的寄存器^Jj^ 寄存器‘ID’/numbei-

X''\^t0-t2 ** x5-x7

t3-t6 ~ x28-x31

—其他寄存器有特殊用途，我们稍后讨论

- 寄存器无类型（C 概念）;正在执行的操作决定如何处理寄存器内容

特殊登记册

最重要的数字是什么?

免责声明：编程中

零寄存器

- 零经常出现在代码中，并且非常有用，以至于它有自己的寄存器!
- 寄存器零（x0 或零）的值始终为 0,不能更改!

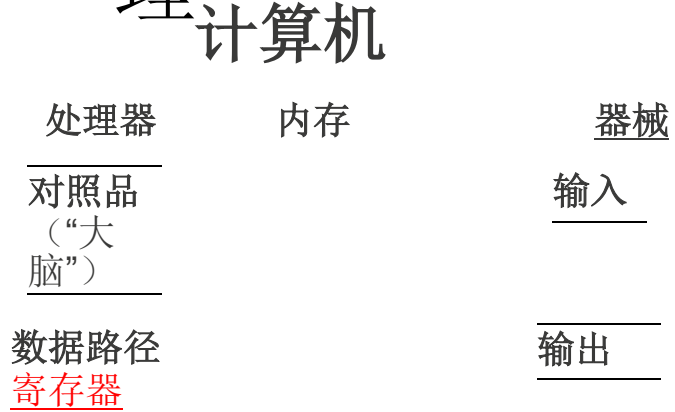
—写入 x0 的任何指令均无效

寄存器 -- 摘要

- 在高级语言中，变量的数量
  仅受可用内存的限制
- ISA 具有固定的少量操作数，称为寄存器
  - 直接内置于硬件中的特殊位置
  - **受益：**寄存器速度极快
    （快于十亿分之一秒）
  - **缺点：**只能在这些预定数量的寄存器上执行操作

在计算机中注册

- 我们开始学习计算机的工作原理
  - —控制
  - —数据路径
  - — 内存
  - —输入
  - —输出



- 寄存器是数据路径的一部分

RISCV 议程

- 背景
- 寄存器
- **装配代码**
- 基本算术指令
- 即时指示
- 数据传输说明
- 控制流程说明
- 换档说明
- 其他有用说明

RISCV 议程

总结

RISCV 指令（1/2）

- 指令语法是严格的：

操作 dst、srcl、src2

—1 个运算符，3 个操作数

- op = 操作名称（“操作员”）
- dst = 寄存器获取结果（“目标”）
- srcl = 操作的第一个寄存器（“源 1”）
- src2 = 第二个操作寄存器（“源 2”）

RISCV 指令（1/2）

- 通过规律性保持硬件的简单性
- 每条指令一个操作，每行最多一条指令
  - 组装说明与 C 相关

操作（=、+、-、*、/、&、|等）

—一定是，因为 C 代码分解成汇编!

—单行 C 可能分解成多行 RISC-V

RISC-V 组件示例

斐波那契序列数量

```
                主峰：   加上      t0, x0, x0
                         addi      t1,x0, 1
                         拉        t3, n
                         长波      t3, 0   3）
                纤维：   贝克      t3, x0, 结束
                         加上       t2,t1, t0
                         毫伏       t0,第 1
                         毫伏       t1,t2
                         addi      t3, t3, -1
                         j         纤
                表面处   addi      a0, x0, 1
                         addi       a1,t0, 0
ecall # 打印整数 ecall
addi a0, x0, 10
ecall # 终止 ecall
```

RISC-V 组件示例

斐波那契序列数量

```
主峰：   添加其   t0, x0, x0
         他 Iw    t1, x0,1
                  t3,n
                  t3,0（t3）

纤维：   贝克     t3, x0, 完成
         加上     t2, t1, t0
         毫伏     t0, t1
         毫伏     t1, t2
         addi     t3, t3, -1
         j        纤维
表面处   addi     a0, x0,1
         addi     a1,t0,0
         召唤     # 打印整数调用
         addi     a0, x0, 10
         召唤     # 终止呼叫
```

各种组件说明

RISC-V 组件示例

斐波那契序列数量

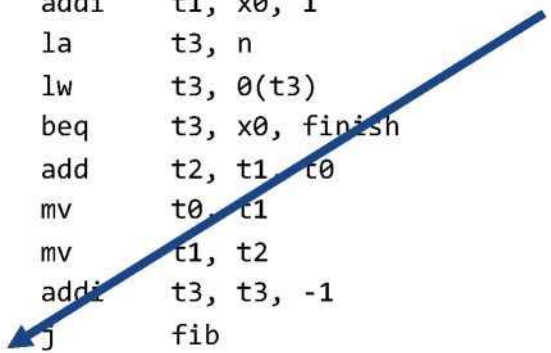
```
主峰：   加上     t0, x0, x0
         addi     t1, x0,1
         拉       t3,
         长波     t3, 0（t3）
纤维：   贝克     t3, x0,完成
         加上     t2, t1,至
         毫伏     t0, 第 1 天
         毫伏     t1, t2
         addi     t3, t3, -1
         j        纤
表面处   addi     a0, x0,1
         addi     a1, t0,0
          ecall # 打印整数 ecall
          addi a0, x0, 10
          ecall # 终止 ecall
```

注释使用 # 符号

RISC-V 组件示例

斐波那契序列主编号：　，

TM,xo,xo



```
        addi    t1, x0, 1
        la      t3, n
        lw      t3, 0(t3)
fib:    beq     t3, x0, finish
        add     t2, t1, t0
        mv      t0, t1
        mv      t1, t2
        addi    t3, t3, -1
        j       fib
finish: addi    a0, x0, 1
        附加，t0,0
        ecall # 打印整数 ecall
        addi a0, x0, 10
        ecall # 终止 ecall
```

标签是标记一段代码的任意名称

我们稍后会再来讨论这些

RISC-V 组件示例

```
# Fibonacci 序列主值：添加 t0,
        4J>•          x0, x0
```

操作 dst、

本说明的作用是什么?

有很多说明

我们将在课堂上讲解指导的类型。

您应该查看绿卡上存在哪些特定命令。

添加，添加	R	ADD（单词）	R[rd] = R[rsl] + R[rs2]
addi,addiw	I	ADD立即数（Word）	R[rd] = R[rsl] + imm
和	R	和	R[rd] = R[rsl] & R[rs2]
安迪	I	与立即数	R[rd] = R[rsl] & imm
auipc	U	将上立即数添加到 PC	R[rd] = PC + {imm, 121）0}
贝克	小型	分支 EQual	如果（R[rsl]-R[rs2）PC=PC+{imm,lb'O}
贝格	小型	分支大于或等于	如果（R[rsl]>=R[rs2）PC=PC+{imm,lb'O}
伯格	小型	分支 > 无符号	如果（R[rsl]>=R[rs2）PC=PC+{imm,lb'O}
钻头	小型	分支小于	如果（R[rsl]<R[rs2）PC=PC+{imm,lb'O}
蓝色的	小型	小于无符号的分支	如果（R[rsl]<R[rs2）PC=PC+{imm,lb'O}
贝恩	小型	分支不相等	if（Rlrsll!=Rlrs2）PC=PC+{imm.lb'O}
裂口	I	环境中断	将控制转移到调试器
召唤	I	环境调用	将控制权转移到操作系统
贾尔	乌杰	跳转和链接	R[rd] = PC+4;PC = PC + {immJb'O}
贾尔	I	跳转和链接寄存器	R[rd] = PC+4;PC = R[rsl]+imm
磅	I	加载字节	R[rd] = {561）M[]（7），M[R[rsl]+imm]（7：0
伊布	I	加载无符号字节	R[rd] = {561>O,M[R[rsl]+imm]（7：O）}
编号	I	加载双字	R[rd] = M[R[rsl]+imm]（63：0）
伊赫	I	加载半字	R[rd] - {481>M[]（15），M[R[rsl]+imm]（15：
伊胡	I	加载无符号半字	R[rd] =
吕伊	U	加载立即数上限	R[rd| = {32b'imm<31>，imm, 121）0}
伊瓦	I	加载字	R[rd]- {321）M[]（31），M[Rrs1]+imm]（31
伊武	I	加载无符号字	R[rd] =
或	R	或	R[rd] = R[rsl]	R[rs2]
ori	I	OR 立即	R[rd] = R[rsl]	imm
某人	S	存储字节	M[R[rsl]+imm]（7：0）= R[rs2]（7：0）
标准差	S	存储双字	M[R[rsl]+imm]（63：0）=
什	S	存储半字	M[R[rsl]+imm]（15：0）=
sll,sllw	R	左移（Word）	R[rd] = R[rsl] « R[rs2]
slli,slliw	I	左移立即数（Word）	R[rd] = R[rsl] « imm
坐	R	设置小于	R[rd] =（R[rsl] < R[rs2]）?1： 0
索尔蒂	I	设置小于立即数	R|rd| =（R[rs 1] < imm）?I： 0
萨尔蒂尤	I	设置 < 无符号立即数	R[rd] =（R[rsl] < imm）?1： 0
原位的	R	设置小于无符号	R[rd] =（R[rsl] <R[rs2]）?1：0
sra,sraw	R	算术右移（Word）	R[rd] = R[rsl] » R[rs2]
srai,sraiw	I	右移算子Imm（Word）	R[rd] = R[rsl] » imm
srl,srlw	R	右移（Word）	R[rd] = R[rsl] » R[rs2]
srli,srliw	I	右移立即数（Word）	R[rd] = R[rsl] » imm
sub,subw	R	SUBtract（Word）	R[rd] = R[rsl] - R[rs2]
SW	S	存储 Word	M[R[rsl]+imm]（31：0）=
异或	R	异或	R[rd] = R[rsl]ÀR[rs2]
肖里	I	XOR 立即数	Rfrdl = Rfrsll A imm

RV64I 基本整数说明，按字母顺序排列

助记符	FMT	姓名	描述（在Verilog中）

RISCV 议程

- 背景
- 寄存器
- 装配代码
- **基本算术指令**
- 即时指示
- 数据传输说明
- 控制流程说明
- 换档说明
- 其他有用说明

RISCV 议程

总结

RISCV 指令示例

这里假设变量 a、b 和 c 分别分配给寄存器 s1、s2 和 s3。

- 整数加法（add）
  —C：a = b + c;
  - RISCV：添加 s1、s2、s3
- 整数减法（减法）
  —C：a = b - c;
  —风险：sub s1、s2、s3

RISCV 指令示例

- 假设 a ^ s0,b ^ s1,c^ s2,d^ s3 和 e ^ s4。将以下 c 语句转换为 RISCV：

```
a=（b+c）-（d+e）;
```

添加 t1,s3,s4 添加 t2,s1,s2 子 s0,t2,t1

说明事项的顺序（必须遵循操作顺序）

使用临时寄存器

绿卡说明

助记符 FMT 名称
`add,addw R ADD（字）`

描述（在Verilog中）
R[rd] - R（rsl] + R[rs2]

1

Verilog（硬件描述）
R：寄存器数组

组装中的操作
（最后使用 w 的操作使用 32 位（RISC-V 中的字），用于大于 32 位的系统）

读取为：
读取 rs1 处的寄存器与 rs2 处的寄存器，将它们相加，然后写入 rd 处的寄存器

RISCV 议程

- 背景
- 寄存器
- 装配代码
- 基本算术指令
- **即时指示**
- 数据传输说明
- 控制流程说明
- 换档说明
- 其他有用说明

总结

即刻

- 数值常量称为中间值
- 中继的独立指令语法：

```
opi dst, src, imm
```

  - 运算名称以“i”结尾，将第二个源寄存器替换为立即数
  - 立即数最多可达 12 位
  - 解释为符号扩展二的补码
- 使用示例：

```
—addi s1, s2, 5 # a=b+5
—addi s3, s3, 1 # c++
```

RISC-V 即时示例

- 假设 a ^ s0,b^ s1.

  将以下 C 语句转换为 RISCV： a =（5+b）-3;

```
附加 1, s1, 5
addi s0, t1, -3
```

- 为什么没有 subi 指令?

RISCV 议程

- 背景
- 寄存器
- 装配代码
- 基本算术指令
- 即时指示
- **数据传输说明**
- 控制流程说明
- 换档说明
- 其他有用说明

RISCV 议程

总结

数据传输

- C 变量映射到寄存器上;
  数组之类的大型数据结构呢?
  — 程序集还可以访问内存
- 但 RISCV 指令只对寄存器进行操作!
- 专用数据传输指令在寄存器和存储器之间移动数据
  —存储： 寄存器至存储器
  — 加载： 寄存器来自存储器

计算机的五大组成部分

- 数据传输指令在寄存器（数据路径）和存储器之间
  — 允许我们在内存中读取和存储操作数

计算机

处理器　　内存　　器械

对照品
（“大

输入

数据路径
寄存器

储存（至）

加载（来自）

数据传输

- 数据传输指令语法：

```
memop reg, off（bAddr）
```

— memop = 操作名称（“操作员”）
— reg = 操作源或目的地的寄存器
— bAddr = 带内存指针的寄存器（“基址”）
— off = 地址偏移（立即），单位为字节（“偏移”）

- 在地址 bAddr+off 处访问内存
- **提醒：** 寄存器保存一个原始数据字（无类型）-确保使用指向有效存储器地址的寄存器（和偏移量）

内存是字节寻址的

- 我们在 C 中看到的最小数据类型是什么？

一个字符，它是字中的一个字节（8 位）最低有效字节

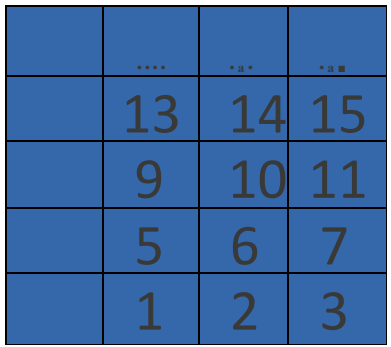
— 所有内容均为 8 位的倍数

（例如，1 个字=4 个字节）

内存地址按字节而不是字索引

字地址相隔 4 个字节

—字地址与第一个字节相同



数据传输说明

—地址必须是 4 的倍数才能“word-aigned”

- 在汇编中不为您执行指针运算 — 必须自己考虑数据大小

- 加载字（lw）
  - 从内存中取出地址 bAddr+off 处的数据，并将其放入 reg
- 存储字（sw）
  - 获取 reg 中的数据并将其存储到地址 bAddr+off 的存储器
- 用法示例：（int array[]的地址-> s3,b-> s2 的值）

C：

数据传输说明

数组[10]=数组[3]+b;

组装：

```
lw t0,12（s3）# t0=A[3]
加 t0,s2,t0 # t0=A[3]+b
sw t0,40（s3）# A[10]=A[3]+b
```

.data • .data 表示数据存储

来源：

O .word、.byte 等

.word 3 用于指向数据的 0 标签
.word 1 • .text 表示代码存储

.word 4 .text main:

la t1,来源

lw t2, 0（t1）Venus 汇编程序指令列表

https://github.com/ThaumicMekanism/venus/wiki/Asse lw t3,4（t1）mbler-

数值可在内存中开始

Directives

内存和可变大小

- 迄今为止：
  —lw reg, off（bAddr）

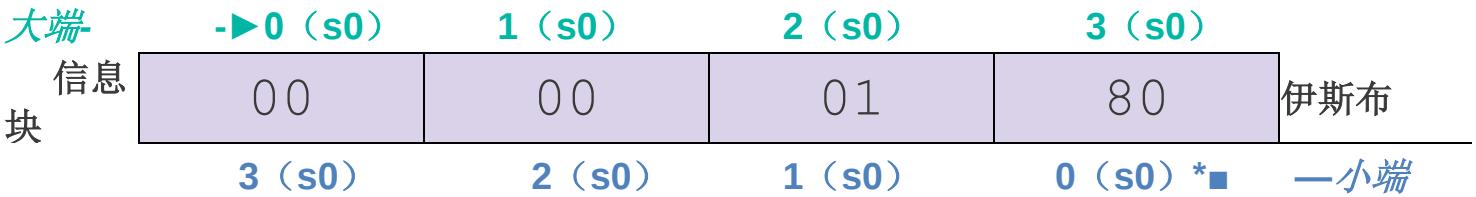
- 字符（1 字节）和短接（有时 2 字节）等呢?

  希望能够使用小于一个单词的内存值进行交互。

字节序

- 大端： 最高有效字节至少是字的地址— 字地址= 最高有效字节的地址
- Little Endian： 字的最小地址的最低有效字节
  — 字地址= 最低有效字节的地址

```
s0 = 0x00000180
```



- RISC-V 是 Little Endian

标志扩展

- 我们希望使用比以前更多的位来表示相同的数字
  - 符号扩展：取最高有效位并将其复制到新位
    - 0b 11 = 0b 1111
  - 零/一焊盘：用 1/0 设置新位。
    - 零点填充：0b 11 = 0b 0011
    - 一个衬垫：0b 11 = 0b 1111
- 所有基本 RISC-V 指令在需要时签署延期
  - 从技术上讲，auipc & lui 会这么做，但他们正在填补高位，所以没有什么可填补的。

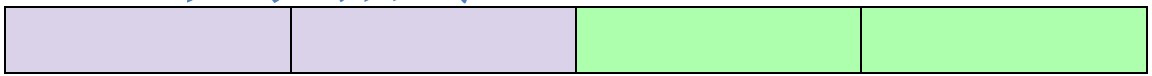
字节指令

- lb/sb 利用寄存器的最低有效字节
  — 在 sb 上，高 24 位被忽略
  — 在 lb 上，高 24 位通过符号扩展填充

- 例如，设 s0 = 0x00000180：

```
磅 s1,1 (s0)      # s1=0x00000001
                  # s2=0xFFFFFF80
磅 s2,0           编号*（s0）=0x00800180
```

半字指令



- lh reg, off（bAddr）“半载”
- sh reg, off（bAddr）“存储一半”

— 在 sh 上，高 16 位被忽略
— 在 lh 上，通过符号扩展填充高 16 位

未签名指令

- lhu reg, off（bAddr）"加载半无符号"
- lbu reg, off（bAddr）“load byte unsigned”

— 在 l（b/h）u 上，高位由零扩展填充

为什么没有 s（h/b）u?为什么没有 lwu?

数据传输绿卡说明

存储字：　M[R[rs1]+imm]（31：0）= R[rs2]（31：0）

内存　　b 地址+ 偏移　　选择位 31-0

存储字节：　M[R[rs1]+imm]（7：0）= R[rs2]（7：0）

数据传输绿卡说明

加载字节：R[rd] = {24’bM[]（7），M[R[Rs1]+imm]（7:0）}（带符号）I u J I \ I

基于最高有效位进
行符号扩展的 24 位
来自内存的字节

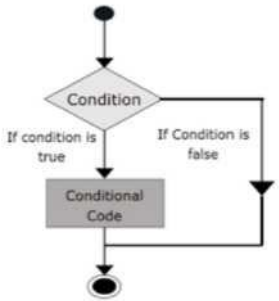
RISCV 议程

- 背景
- 寄存器
- 装配代码
- 基本算术指令
- 即时指示
- 数据传输说明
- **控制流程说明**
- 换档说明
- 其他有用说明
- 总结

计算机决策

在 C 中，我们有控制流

— *比较/逻辑语句的结果决定了要执行的代码块*



在 RISCV 中，我们不能定义代码块;

我们只有标签

– 由后跟冒号的文本定义（例如，main：），并参考随后的说明

— 通过跳转到标签生成控制流

—C 也有标签，但它们被认为是不好

决策说明

的样式。

不要使用 goto!

- 相等时的分支（beq）
  —beq reg1,reg2,标签
  — 如果 reg1 中的值= reg2 中的值，则转到标签
  —否则转至下一指令
- 分支，如果不相等（bne）
  —bne reg1,reg2,标签

决策说明

— 如果 reg1 中的值^ reg2 中的值，请转到标签

- 跳跃（j）
  - —j 标签
  - — 无条件跳转到标签

分解 If Else

C 代码：

```
if（i==j）{
  a = b /* 然后 */
} else {
  a = -b /* else */
}
```

英语：

- 如果为 TRUE,则执行 THEN 块
- 如果为 FALSE,则执行 ELSE 块

RISCV（beq）：

```
# i-s0,j-s1
# a—s2, b—s3 beq
s0,s1,???
???: J 无需此标签
子 s2,x0,s3
j 端
???: B * * •
添加 s2,s3,x0
结束：
```

C 代码：

分解 If Else

```
if（i==j）{
  a = b /* 然后 */
} else {
  a = -b /* else */
}
```

英语：

- 如果为 TRUE,则执行 THEN 块
- 如果为 FALSE,则执行 ELSE 块

RISCV

（bne）：

```
# i-s0,j-s1
# a—s2,b—s3 bne
s0,s1,否则则：
添加 s2, s3, x0
j end else:
子 s2、x0、s3 结
束：
```

在其他条件下分支
不等于

- 分支小于（blt）
  - blt reg1,reg2,标签
  - 如果 reg1 中的值< reg2 中的值，则转到标签
- 分支大于或等于（bge）
  - bge reg1,reg2,标签
  - 如果 reg1 中的值>= reg2 中的值，则转到标签

RISC 原理：

- 如果可以交换参数并使用“branch less than”，为什么要创建“大于”的分支

分解 If Else

C 代码：

```
如果（i<j）{
  a = b /* 然后 */
} else {
  a = -b /* else */
}
```

英语：

- 如果为 TRUE,则执行 THEN 块
- 如果为 FALSE,则执行 ELSE 块

RISCV（???）：

```
# i-s0,j-s1
# a—s2,b—s3

???s0,s1,否则：
添加 s2, s3, x0 j end
else:
子 s2、x0、s3 结束：
```

RISCV 中的环路

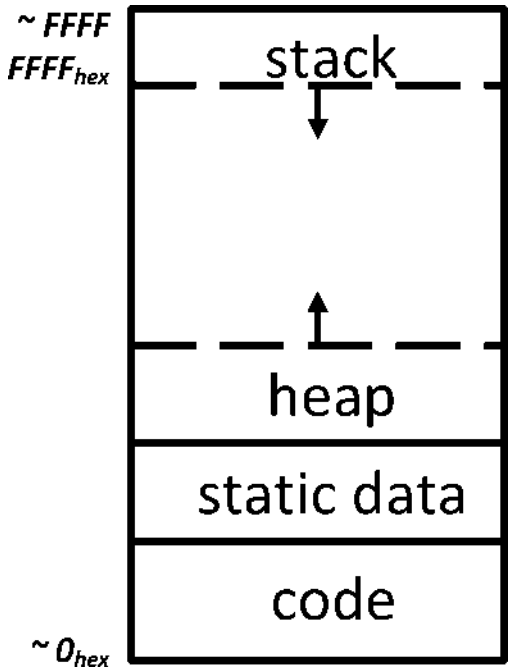
- C 中有三种类型的循环：
  - while, do... what, and for
  - —每个都可以改写为其他两个中的任何一个，
    所以同样的决策概念
- 这些也可以通过分支指令创建
- **关键概念：**虽然在 RISCV 中写循环有多种方法，但决策的关键是条件分支

指令地址
程序计数器

- 分支和跳转通过修改程序计数器（PC）来更改执行流程

- PC 是一个特殊寄存器，包含正在执行的代码的当前地址— 不能作为正常 32 个寄存器的一部分进行访问

- 指令作为数据存储在内存中，并有地址!
  — 召回：代码部分
  —标签转换为指令地址
  —本周晚些时候对此进行详细介绍
- PC 跟踪当前指令在内存中的位置



指令地址控制流程绿卡说明

分支相等（beq）：如果（R[rs1] == R[rs2]）

PC =

PC+{imm,1b’0} I I

如果寄存器相等：
跳转到立即数，前加一位 0

（使 PC 为偶数）

立即数计算为标签的偏移量

RISCV 议程

- 背景
- 寄存器
- 装配代码
- 基本算术指令
- 即时指示
- 数据传输说明
- 控制流程说明
- **换档说明**
- 其他有用说明

总结

换档说明

- 在二进制中，将无符号数左移等于乘以相应的 2 的幂。
  — 换档操作更快
  —右移/分割时不起作用
- *逻辑移位*：移位时添加零
- *算术移位*：换档时扩展符号
  —仅适用于右移（保留符号）
- 按立即数或寄存器中的值移位

换档说明

指令名称	RISCV
逻辑左移	sll s1,s2,s3
逻辑 I 毫米左移	索利 s1,s2,imm
逻辑右移	srl s1,s2,s3
逻辑 Imm 右移	斯里 s1,s2,imm
算术右移	sra s1,s2,s3
算术右移 Imm	斯拉 s1,s2,imm

- 使用立即数时，只有值 0-31 是实用的
- 使用变量时，仅使用最低的 5 位（读取为无符号）

```
#  对移位指令 addi t0,x0，-256 #
t0=0xFFFFFF00 的示例调用
```

换档说明

```
slli s0,t0,3 # s0=0xFFFFF800
srli s1,t0,8 # s1=0x00FFFFFF
srai s2,t0,8 # s2=0xFFFFFFFF

addi t1,x0, -22 # t1=0xFFFFFFEA
                        #低水平 5：0b01010
sll s3,t0,t1 # s3=0xFFFC0000
# 与 slli s3,t0,10 相同
```

- 示例 1：

使用 Iw 的 lbu 数量：Ibu s1,1（s0）

```
Iw s1,0（s0）# 获取信息
```

换档说明

```
li t0,0x0000FF00 # 加载位掩码
和 s1,s1,t0 # 获取第 2 个字节
srli s1,s1,8 # 移入最低
```

- 示例 2：

```
# 使用 sw 的 sb: sb s1,3（s0）
Iw t0,0（s0）# 获取当前单词
li t1, 0x00FFFFFF # 加载位掩码
以及 t0,t0,t1#顶部为零的字节
slli t1,s1,24 # 移入最高
或 t0,t0,t1 # 组合
```

换档说明

```
sw t0,0（s0）# 存储返回
```

RISCV 议程

- 背景
- 寄存器
- 装配代码
- 基本算术指令
- 即时指示
- 数据传输说明
- 控制流程说明
- 换档说明
- **其他有用说明**

RISCV 议程

总结

RISCV 算术指令乘法扩展

- 乘法（mul 和 mulh）
  - —mul dst、srcl、src2
  - -mulh dst, srcl, src2
  - — src1*src2：低 32 位至 mul,高 32 位至 mulh
- 分部（div）
  - —div dst、srcl、src2
  - — rem dst、srcl、src2
  - — src1/src2：商通过 div,余数通过 rem

RISCV 位指令

注：a^sl、b^s2、c^s3

说明	C	RISCV
和	a=b&c;	以及 S1,S2,S3
和即时	a = b & 0x1;	和 s1,s2,0x1
或	a=b|c;	或 s1,s2,s3
或立即	a = b | 0x5;	或 s1,s2,0x5
独占或	a=bAc;	s1,s2,s3 异或
独占或即时	a = b A 0xF;	xori s1,s2,0xF

比较指令

- 设置小于（slt）
  - slt dst, reg1, reg2
  - 如果 regl 中的值< reg2 中的值，则 dst = 1,否则为 0
- 设置小于立即数（slti）
  - `slti dst,reg1,imm`
  - 如果 regl 中的值< imm,则 dst = 1,否则为 0

另一个 If Else 设计

C 代码：

```
如果（i<j）{
  a = b /* 然后 */
} else {
  a = -b /* else */
}
```

英语：

- 如果为 TRUE,则执行 THEN 块
- 如果为 FALSE,则执行 ELSE 块

RISCV：

```
# i-s0,j-s1
# a—s2,b—s3

slt t0 s0 s1 beq t0,x0,否则
则：
   添加 s2,s3,x0
   j 端
   否则：
子 s2、x0、s3 结束：
```

环境调用

- `ecall` 是应用程序与操作系统交互的一种方式
- 寄存器 a0 中的值提供给执行特殊功能的操作系统
  - 打印值
  - 退出程序
  - 为程序分配更多内存

venus 支持的 ecall 值的列表：

https://github.com/ThaumicMekanism/venus/wiki/Environmental-Calls

RISCV 议程

- 背景
- 寄存器
- 装配代码
- 基本算术指令
- 即时指示
- 数据传输说明
- 控制流程说明
- 换档说明
- 其他有用说明

总结

总结

- 计算机理解其指令集体系结构（ISA）的指令
- RISC 设计原理
  —越小越快，保持简单
- RISC-V 寄存器：s0-s11、t0-t6、x0
- RISC-V 指令
  —算术运算：add、sub、addi
  —数据传输： `lw, sw`
    - 内存为字节寻址
  —控制流程：beq, bne, j