汇编语言

@M了个J

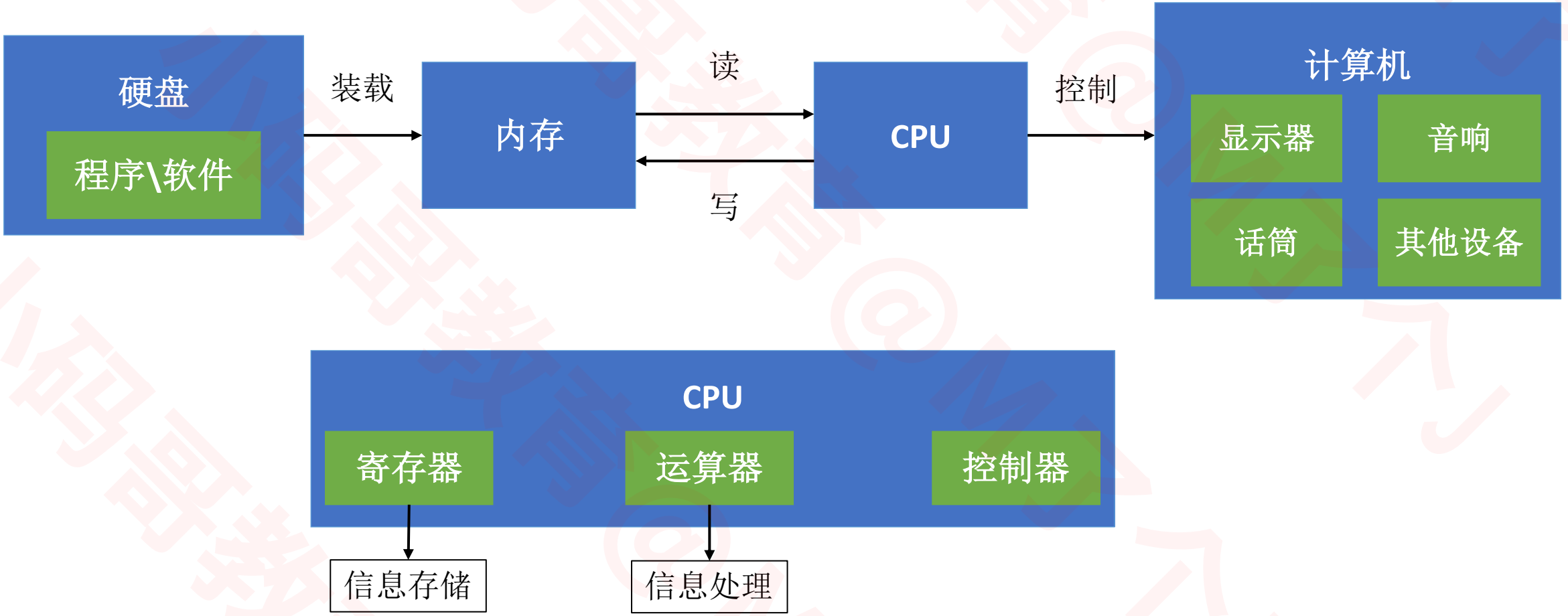
https://github.com/CoderMJLee

http://cnblogs.com/mjios

码拉松

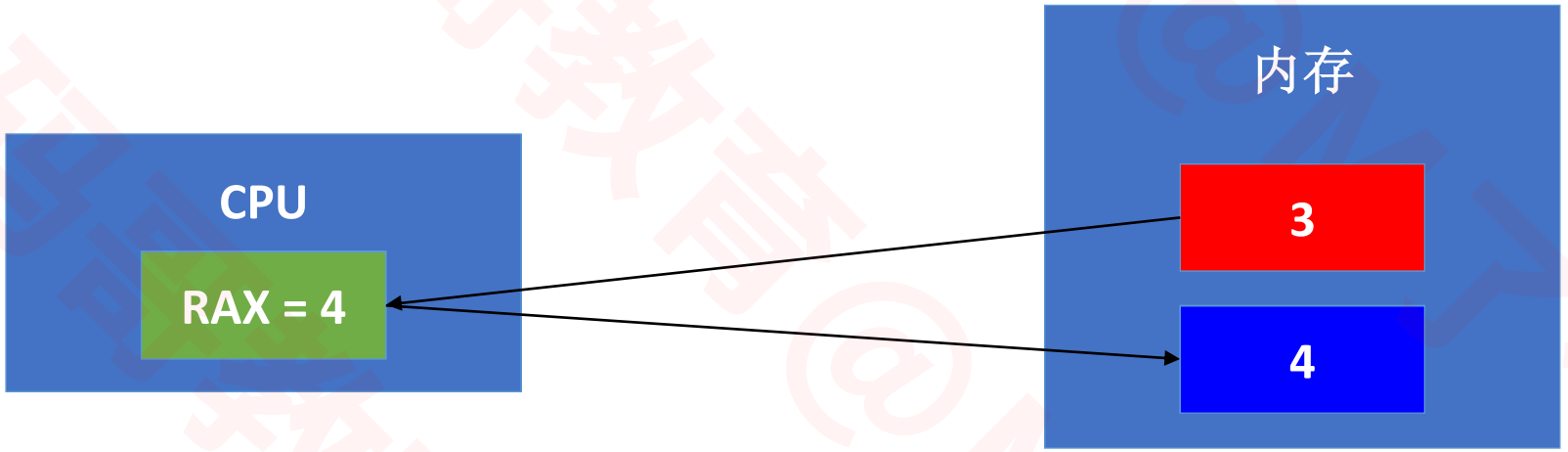
程序的本质

- 软件\程序的执行过程



寄存器与内存

- 通常，CPU会先将内存中的数据存储到寄存器中，然后再对寄存器中的数据进行运算
- 假设内存中有块红色内存空间的值是3，现在想把它的值加1，并将结果存储到蓝色内存空间
  - CPU首先会将红色内存空间的值放到rax寄存器中：movq 红色内存空间, %rax
  - 然后让rax寄存器与1相加：addq $0x1, %rax
  - 最后将值赋值给内存空间：movq %rax, 蓝色内存空间

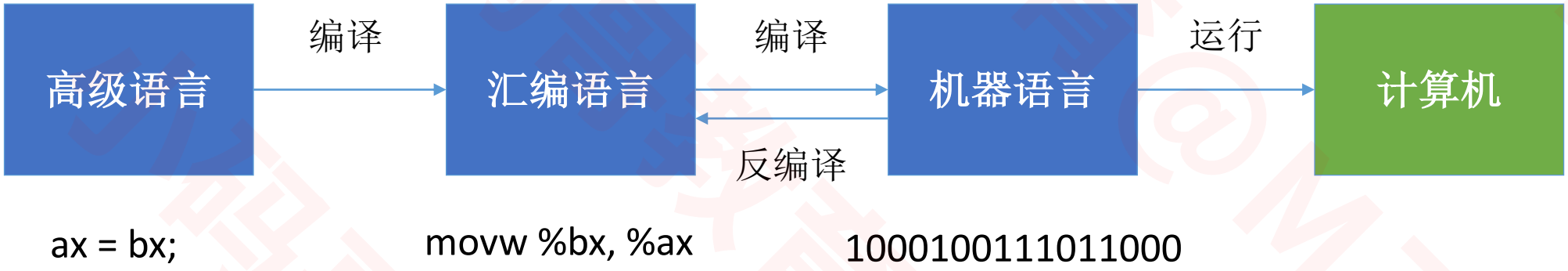


编程语言的发展

- 机器语言
  - 由0和1组成
- 汇编语言（Assembly Language）
  - 用符号代替了0和1，比机器语言便于阅读和记忆
- 高级语言
  - C\C++\Java\JavaScript\Python等，更接近人类自然语言

- 操作：将寄存器BX的内容送入寄存器AX
  - 机器语言：1000100111011000
  - 汇编语言：movw %bx, %ax
  - 高级语言：ax = bx;

编程语言的发展



- 汇编语言与机器语言一一对应，每一条机器指令都有与之对应的汇编指令
- 汇编语言可以通过编译得到机器语言，机器语言可以通过反汇编得到汇编语言
- 高级语言可以通过编译得到汇编语言\机器语言，但汇编语言\机器语言几乎不可能还原成高级语言

汇编语言的种类

- 汇编语言的种类
  - 8086汇编(16bit)
  - x86汇编(32bit)
  - x64汇编(64bit)
  - ARM汇编(嵌入式、移动设备)
  - ……

- x86、x64汇编根据编译器的不同，有2种书写格式
  - Intel：Windows派系
  - AT&T ：Unix派系

- 作为iOS开发工程师，最主要的汇编语言是
  - AT&T汇编 -> iOS模拟器
  - ARM汇编 -> iOS真机设备

小码哥教育 SEEMYGO

常见汇编指令

项目	AT&T	Intel	说明
寄存器命名	%rax	rax	
操作数顺序	movq %rax, %rdx	mov rdx, rax	将rax的值赋值给rdx
常数\立即数	movq $3, %rax movq $0x10, %rax	mov rax, 3 mov rax, 0x10	将3赋值给rax 将0x10赋值给rax
内存赋值	movq $0xa, 0x1ff7(%rip)	mov qword ptr [rip+0x1ff7], 0xa	将0xa赋值给地址为rip + 0x1ff7的内存空间
取内存地址	leaq -0x18(%rbp), %rax	lea rax, [rbp – 0x18]	将rbp – 0x18这个地址值赋值给rax
jmp指令	jmp *%rdx jmp 0x4001002 jmp *(%rax)	jmp rdx jmp 0x4001002 jmp [rax]	call和jmp写法类似

操作数长度	movl %eax, %edx
	movb $0x10, %al
	leaw 0x10(%dx), %ax
		mov edx, eax
		mov al, 0x10
		lea ax, [dx + 0x10]
			b = byte (8-bit)
			s = short (16-bit integer or 32-bit floating point)
			w = word (16-bit)
			l = long (32-bit integer or 64-bit floating point)
			q = quad (64 bit)
			t = ten bytes (80-bit floating point)

小码哥教育 SEEMYGO

寄存器

■ 有16个常用寄存器

- rax、rbx、rcx 、rdx、rsi、rdi、rbp、rsp
- r8、r9、r10、r11、r12、r13、r14、r15

■ 寄存器的具体用途

- rax、rdx常作为函数返回值使用
- rdi、rsi、rdx、rcx、r8、r9等寄存器常用于存放函数参数
- rsp、rbp用于栈操作
- rip作为指令指针
  - ✓ 存储着CPU下一条要执行的指令的地址
  - ✓ 一旦CPU读取一条指令，rip会自动指向下一条指令（存储下一条指令的地址）

小码哥教育 SEEMYGO

寄存器

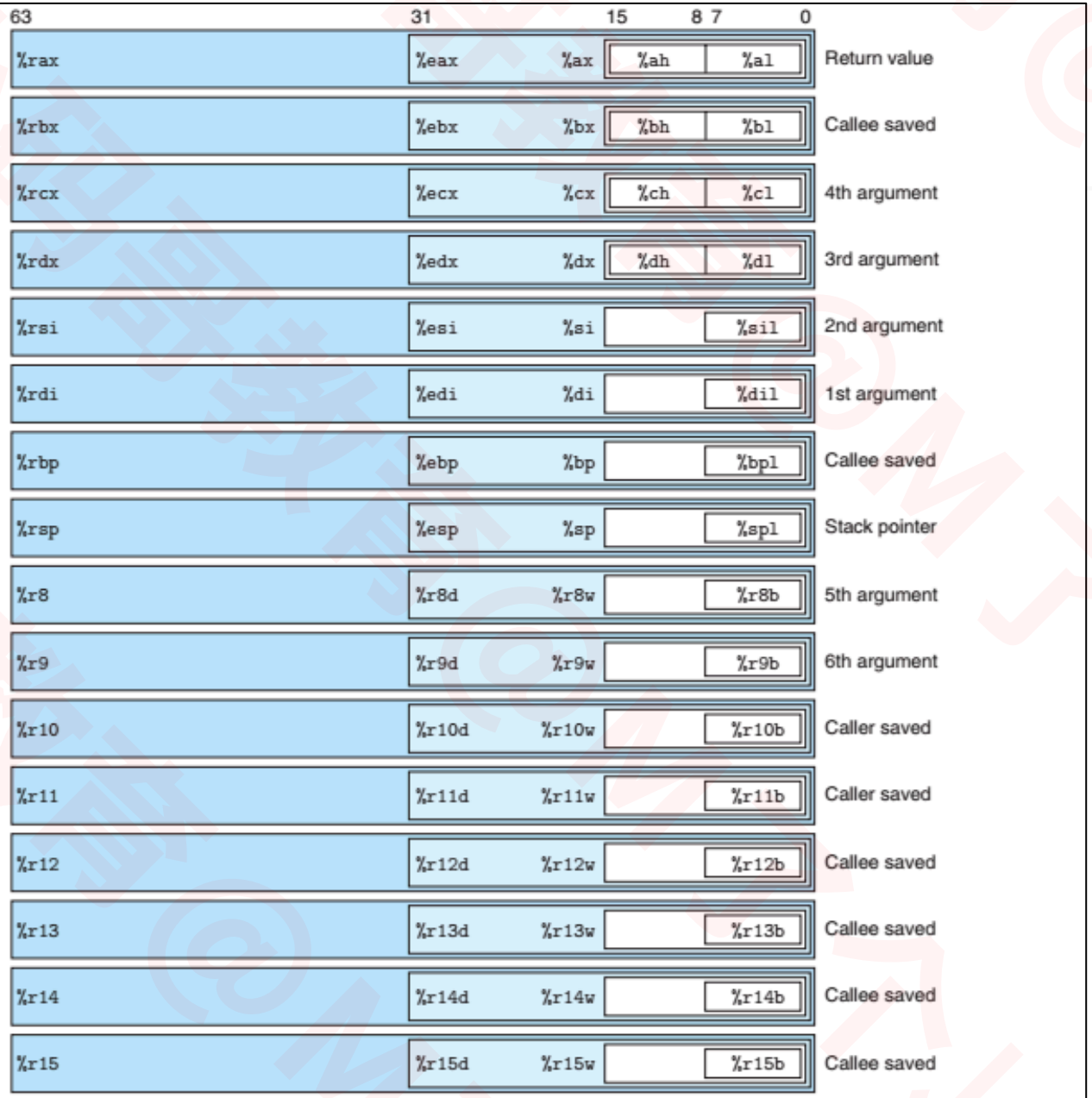


Figure 3.35 **Integer registers.** The existing eight registers are extended to 64-bit versions, and eight new registers are added. Each register can be accessed as either 8 bits (byte), 16 bits (word), 32 bits (double word), or 64 bits (quad word).

lldb常用指令

- 读取寄存器的值
  - register read/格式
  - register read/x

- 修改寄存器的值
  - register write 寄存器名称 数值
  - register write rax 0

- 读取内存中的值
  - x/数量-格式-字节大小 内存地址
  - x/3xw 0x0000010

- 修改内存中的值
  - memory write 内存地址 数值
  - memory write 0x0000010 10

- 格式
  - x是16进制，f是浮点，d是十进制

- 字节大小
  - b – byte 1字节
  - h – half word 2字节
  - w – word 4字节
  - g – giant word 8字节

- expression 表达式
  - 可以简写：expr 表达式
  - expression $rax
  - expression $rax = 1

- po 表达式
- print 表达式
  - po/x $rax
  - po (int)$rax

lldb常用指令

- thread step-over、next、n
  - 单步运行，把子函数当做整体一步执行（源码级别）

- thread step-in、step、s
  - 单步运行，遇到子函数会进入子函数（源码级别）

- thread step-inst-over、nexti、ni
  - 单步运行，把子函数当做整体一步执行（汇编级别）

- thread step-inst、stepi、si
  - 单步运行，遇到子函数会进入子函数（汇编级别）

- thread step-out、finish
  - 直接执行完当前函数的所有代码，返回到上一个函数（遇到断点会卡住）

小码哥教育 SEEMYGO

规律

- 内存地址格式为：`0x4bdc(%rip)`，一般是全局变量，全局区（数据段）
- 内存地址格式为：`-0x78(%rbp)`，一般是局部变量，栈空间
- 内存地址格式为：`0x10(%rax)`，一般是堆空间