计算机系统

4. 程序的机器级表示-基础

华东师范大学 数据科学与工程学院

2021年09月22日

钱卫宁

wnqian@dase.ecnu.edu.cn

体系结构（Architecture）

A.k.a ISA: instruction set architecture

编写正确的机器/汇编代码所需了解的处理器设计部分

包含：指令集规范，寄存器等

- 机器代码：处理器能够执行的字节级程序；
- 汇编代码：机器代码的文本表示。

重要的 ISA

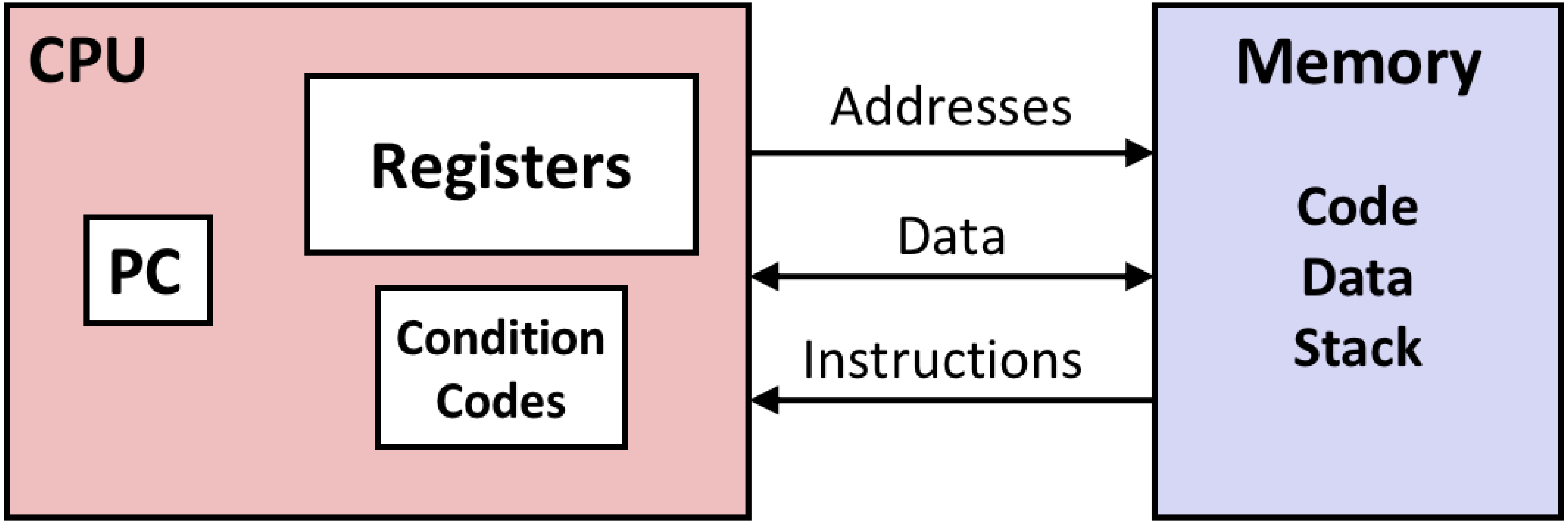
- Intel: x86, IA32, Itanium, x86-64
- ARM: Used in almost all mobile phones
- RISC V: New open-source ISA

微体系结构（Microarchitecture）

体系结构的实现

包含：cache（缓存）大小、主频等

汇编程序所面对的抽象计算机



汇编语言中的数据类型

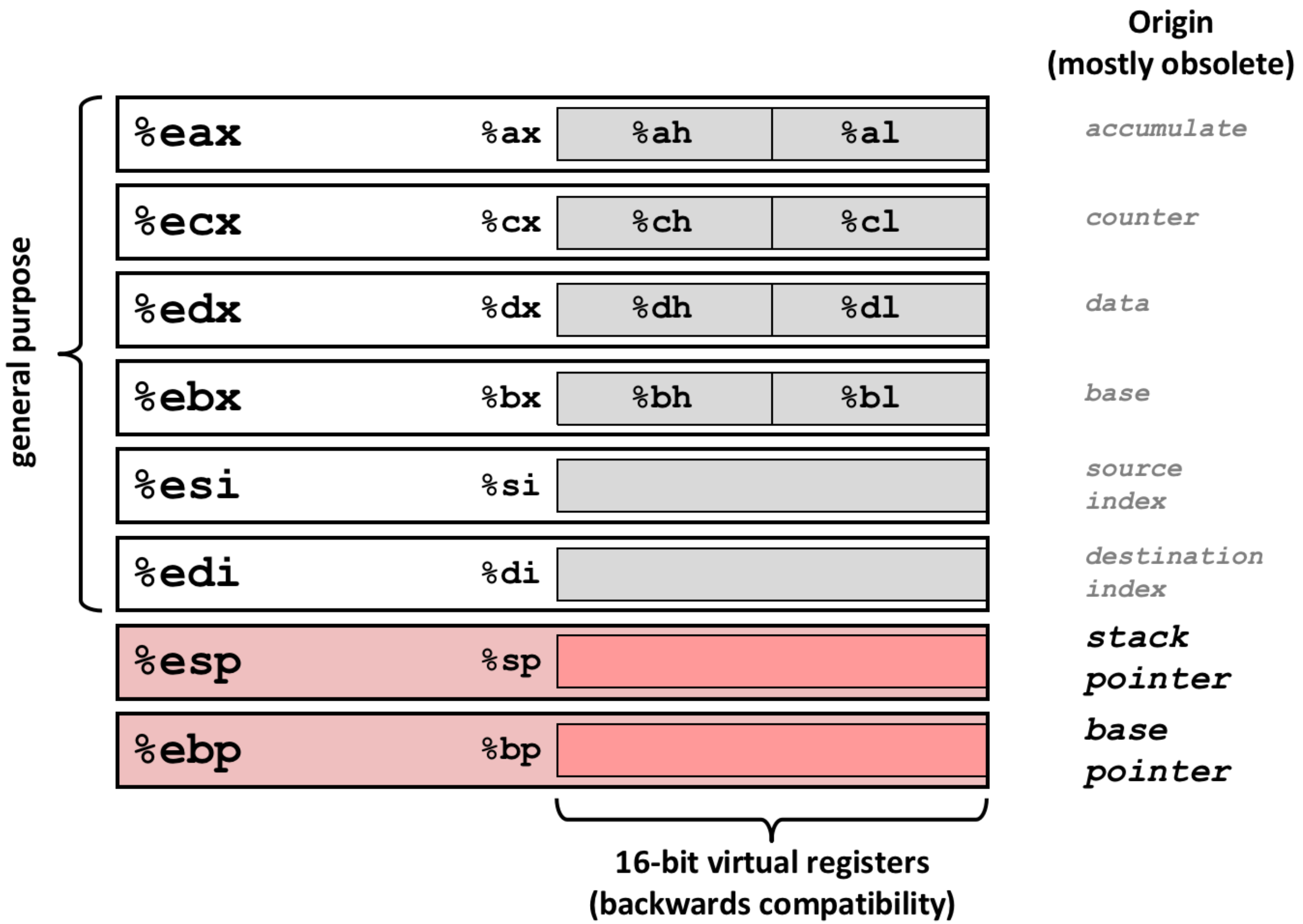
- “Integer” data of 1, 2, 4, or 8 bytes
  - Data values
  - Addresses (untyped pointers)
- Floating point data of 4, 8, or 10 bytes
- (SIMD vector data types of 8, 16, 32 or 64 bytes)
- Code: Byte sequences encoding series of instructions
- 汇编语言不支持像数组、结构这样的复合数据类型
  - 但能够分配内存中连续的字节

x86-64 整数寄存器

%rax	%eax	%r8	%r8d
%rbx	%ebx	%r9	%r9d
%rcx	%ecx	%r10	%r10d
%rdx	%edx	%r11	%r11d
%rsi	%esi	%r12	%r12d
%rdi	%edi	%r13	%r13d
%rsp	%esp	%r14	%r14d
%rbp	%ebp	%r15	%r15d

- Can reference low-order 4 bytes (also low-order 1 & 2 bytes)
- Not part of memory (or cache)

IA32 整数寄存器



Origin
(mostly obsolete)
general purpose
%eax
%ax
%ah
%al
accumulate
%ecx
%cx
%ch
%cl
counter
%edx
%dx
%dh
%dl
data
%ebx
%bx
%bh
%bl
base
%esi
%si
source index
%edi
%di
destination index
%esp
%sp
stack pointer
%ebp
%bp
base pointer
16-bit virtual registers
(backwards compatibility)

汇编指令

- 移数：在内存和寄存器间移动数据
  - 从内存加载（load）数据，放入寄存器
  - 将数据从寄存器存入（store）内存
- 运算：对在寄存器或内存中的数据进行算术运算
- 控制：修改下一条执行的指令
  - 无条件跳转
  - 条件分枝
  - 间接分枝

移数

`movq Source, Destination`

- 直接数：整数常数，以 `$` 开始，格式类似于C语言，如 `$0x400,$-533`
  - 1, 2或4字节
- 寄存器：16个整数寄存器之一（`%rsp` 保留不用）
- 内存：通过寄存器指明的某地址开始的8个连续字节

  （`movq` 中 `q` 对应于8字节）

注意：Intel手册中的格式不同，为： `mov Destination, Source`

移数的不同类型操作数组合

	Source	Dest	Src,Dest	C Analog
`movq`	*Imm*	*Reg*	`movq $0x4,%rax`	`temp = 0x4;`
		Mem	`movq $-147,(%rax)`	`*p = -147;`
	Reg	*Reg*	`movq %rax,%rdx`	`temp2 = temp1;`
		Mem	`movq %rax,(%rdx)`	`*p = temp;`
	Mem	*Reg*	`movq (%rax),%rdx`	`temp = *p;`

Cannot do memory-memory transfer with a single instruction

寻址模式

Type	Form	Operand value	Name
Immediate	$\$Imm$	Imm	Immediate
Register	r_a	$R[r_a]$	Register
Memory	Imm	$M[Imm]$	Absolute
Memory	(r_a)	$M[R[r_a]]$	Indirect
Memory	$Imm(r_b)$	$M[Imm + R[r_b]]$	Base + displacement
Memory	(r_b,r_i)	$M[R[r_b] + R[r_i]]$	Indexed
Memory	$Imm(r_b,r_i)$	$M[Imm + R[r_b] + R[r_i]]$	Indexed
Memory	$(,r_i,s)$	$M[R[r_i] \cdot s]$	Scaled indexed
Memory	$Imm(,r_i,s)$	$M[Imm + R[r_i] \cdot s]$	Scaled indexed
Memory	(r_b,r_i,s)	$M[R[r_b] + R[r_i] \cdot s]$	Scaled indexed
Memory	$Imm(r_b,r_i,s)$	$M[Imm + R[r_b] + R[r_i] \cdot s]$	Scaled indexed

Figure 3.3 **Operand forms.** Operands can denote immediate (constant) values, register values, or values from memory. The scaling factor s must be either 1, 2, 4, or 8.

示例

```
void swap
   (long *xp, long *yp)
{
  long t0 = *xp;
  long t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

```
swap:
   movq    (%rdi), %rax
   movq    (%rsi), %rdx
   movq    %rdx, (%rdi)
   movq    %rax, (%rsi)
   ret
```

算术和逻辑运算指令

Instruction		Effect	Description
`leaq`	S, D	$D \leftarrow \&S$	Load effective address
INC	D	$D \leftarrow D+1$	Increment
DEC	D	$D \leftarrow D-1$	Decrement
NEG	D	$D \leftarrow -D$	Negate
NOT	D	$D \leftarrow \sim D$	Complement
ADD	S, D	$D \leftarrow D + S$	Add
SUB	S, D	$D \leftarrow D - S$	Subtract
IMUL	S, D	$D \leftarrow D * S$	Multiply
XOR	S, D	$D \leftarrow D\ \hat{}\ S$	Exclusive-or
OR	S, D	$D \leftarrow D \mid S$	Or
AND	S, D	$D \leftarrow D \,\&\, S$	And
SAL	k, D	$D \leftarrow D << k$	Left shift
SHL	k, D	$D \leftarrow D << k$	Left shift (same as SAL)
SAR	k, D	$D \leftarrow D >>_A k$	Arithmetic right shift
SHR	k, D	$D \leftarrow D >>_L k$	Logical right shift

Figure 3.10 **Integer arithmetic operations.** The load effective address (`leaq`) instruction is commonly used to perform simple arithmetic. The remaining ones are more standard unary or binary operations. We use the notation $>>_A$ and $>>_L$ to denote arithmetic and logical right shift, respectively. Note the nonintuitive ordering of the operands with ATT-format assembly code.

算术和逻辑运算指令

无符号和有符号运算的指令无区别

一条有点特殊的指令

`leaq Src, Dst`

- `Src` 为寻址表达式
- 设置 `Dst` 为 `Src` 所指的地址
- 并不真的访问地址所在的内存（`p=&x[i];`）
- 常用于计算：$x + k \times y, k = 1, 2, 4, 8$
- 例如：$x * 12$ 对应于：

```
leaq (%rdi,%rdi,2), %rax  # t = x+2*x
salq $2, %rax             # return t<<2
```

示例

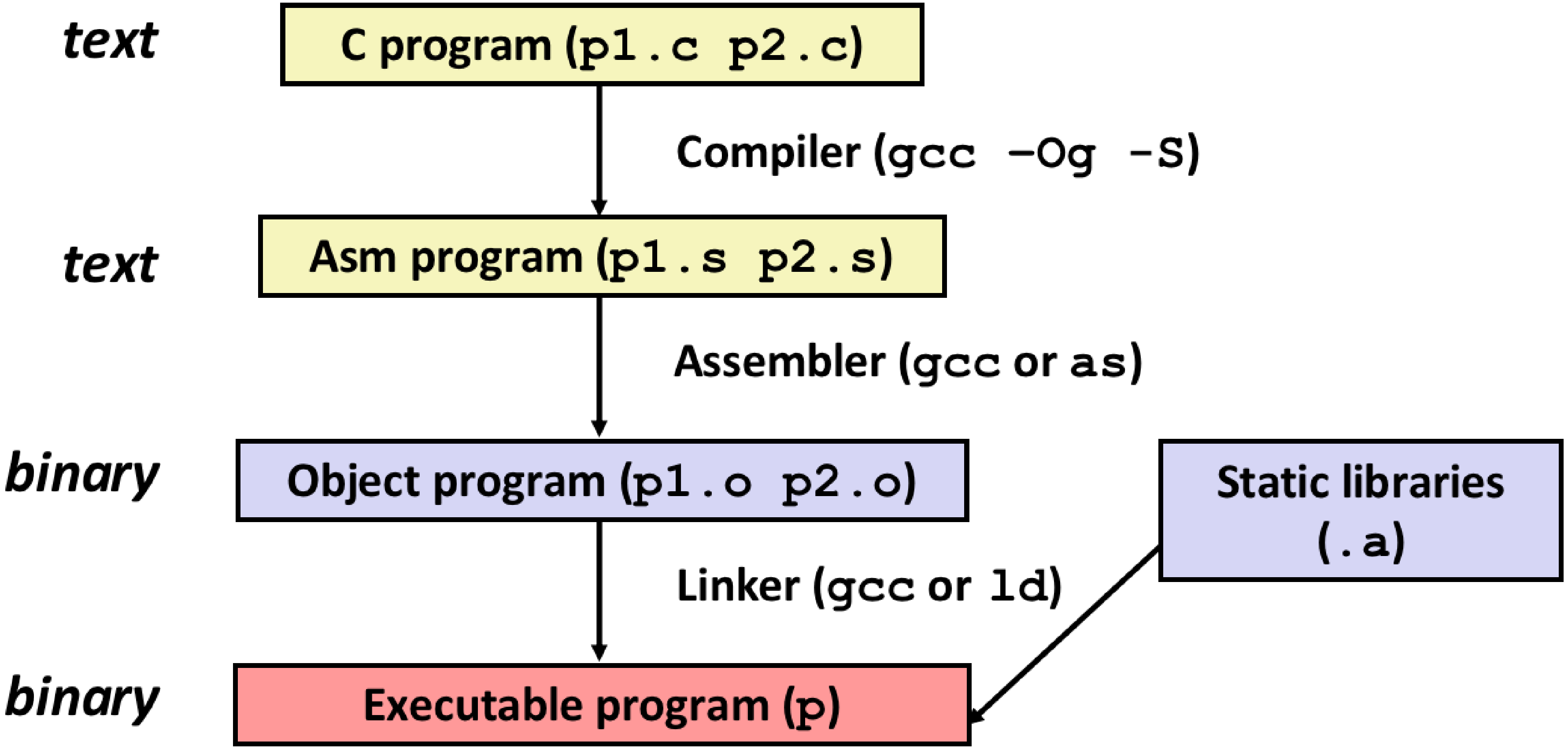
```
long arith
(long x, long y, long z)
{
  long t1 = x+y;
  long t2 = z+t1;
  long t3 = x+4;
  long t4 = y * 48;
  long t5 = t3 + t4;
  long rval = t2 * t5;
  return rval;
}
```

Register	Use(s)
%rdi	Argument **x**
%rsi	Argument **y**
%rdx	Argument **z**, **t4**
%rax	**t1**, **t2**, **rval**
%rcx	**t5**

```
arith:
  leaq    (%rdi,%rsi), %rax    # t1
  addq    %rdx, %rax           # t2
  leaq    (%rsi,%rsi,2), %rdx
  salq    $4, %rdx             # t4
  leaq    4(%rdi,%rdx), %rcx   # t5
  imulq   %rcx, %rax           # rval
  ret
```

从源文件到目标文件

- Code in files `p1.c p2.c`
- Compile with command: `gcc –Og p1.c p2.c -o p`
  - Use basic optimizations (`-Og`) [New to recent versions of GCC]
  - Put resulting binary in file `p`



预习要求

阅读至3.6结束

抽时间仔细/反复阅读第一章