Lesson 5
Machine Prog 3
Processor Arch 1

ICS Seminar #9

张龄心

Oct 18, 2023

指针

- &lvalue用于取指针(lvalue: 左值)

  *p用于解引用(dereference, 读取指针指向的值)

- 左值和右值的定义 (回忆: 程设)
  - 简单粗暴版: 等号左边是左值, 右边是右值
  - 功能角度: 可以取地址且有名字的是左值; 不能取地址且没有名字的是右值

  例: a = 3; a = b + c;

指针运算

- 若p为T *，p + 1即在内存地址上+sizeof(T)
  - void *: 相当于在内存地址上 + 1 (和char*指针的行为相同)
- 指针不能相加
- T * 类型指针相减 = 地址之差 / sizeof (T)
- 不同类型指针不能相减, 会报错

```
void *p = foo();       // 假设p != NULL
int *new_p1 = (int *)p + 7 ;      // new_p1是 int 指针，其值 = p + 28
int *new_p2 = (int *)(p + 7)      // new_p2 是一个 int 指针，其值 = p + 7
```

复习: 复杂类型的阅读

- 从变量名开始看
- 向右读, 直到遇到圆括号时, 转为向左读. 再次遇到圆括号时, 再次转向
- 即, 每次遇到圆括号时就转向

- int a[10]
- int *a[10]
- int (*a)[10]
- int *(*(*a)(int))[10]
- int *(*(*arr[5])())();
- char (*(*x())[])();
- char (*(*x[3])())[5];

例

	sizeof(A)	What is A?
`int *A[3];`		
`int *(A[3]);`		
`int (*A)[3];`		
`int (*A[3]);`		
`int (*A[3])();`		
`int (*A[3])[5];`		
`int (*(A[3])())[5];`		

例

	sizeof(A)	What is A?
`int *A[3];`	24	array[3] of pointer to int
`int *(A[3]);`	24	array[3] of pointer to int
`int (*A)[3];`	8	pointer to array[3] of int
`int (*A[3]);`	24	array[3] of pointer to int
`int (*A[3])();`	24	array[3] of pointer to function returning int
`int (*A[3])[5];`	24	array[3] of pointer to array[5] of int
`int (*(*A[3])())[5];`	24	array[3] of pointer to function returning pointer to array[5] of int

高维数组

- 高维数组实际上是低维数组的嵌套
  - 例: int[5][6] 实际上是 5 个 int[6] 依次排列 (可以推广到任意维数组)
- 关于数组名与 sizeof
  - int A[5][5]; // A 是一个 5*5 的 int 数组，sizeof (A) == 100
  - 但在做指针运算或解引用时，A 会被**隐式转换为一个指向 int[5] 的指针**
  - 故sizeof (A + 1) == 8
- • *A 是一个 5 个 int 的数组，sizeof (*A) == 20
- • 同上，在做指针运算或解引用时，*A 被隐式转换为一个指向 int 的指
- 针，sizeof (*A + 1) == 8

例题

7. 有A的定义：int A[3][2] = {{1,2}, {3,3}, {2,1}};
那么A[2]的值为：

A. &A+16　　B. A+16　　C. *A+4　　D. *A+2

例题

7. 有A的定义：int A[3][2] = {{1,2}, {3,3}, {2,1}};
那么A[2]的值为：

A. &A+16　　B. A+16　　C. *A+4　　D. *A+2

答案：C

解析：参见书P177页表格，机器在计算指针与常数的运算时，会将常数乘以指针指向的元素大小。&A常数扩大的倍数为sizeof(A[3][2]) = 3*2*4; A常数扩大的倍数为sizeof(A[0])=2*4; *A常数扩大的倍数为sizeof(int) = 4。正确的答案应为A+2或*A+4，故应选择C。

例题

定义：

```
int a[3][4] = { {0,1,2,3},{4,5,6,7},{8,9,10,11} };
int(*d)[4] = a + 1;
```

已知 a 的地址为 0x012FFE34，请计算以下表达式的结果：

1. d[1]
2. *d[1]
3. (*d)[1]
4. *(*d + 1)
5. a[2]
6. *a[2]
7. (*a)[2]
8. *(*a + 2)

例题

定义：

```
int a[3][4] = { {0,1,2,3},{4,5,6,7},{8,9,10,11} };
int(*d)[4] = a + 1;
```

已知 a 的地址为 0x012FFE34，请计算以下表达式的结果：

1. d[1]	1. 012FFE54
2. *d[1]	2. 8
3. (*d)[1]	3. 5
4. *(*d + 1)	4. 5
5. a[2]	5. 012FFE54
6. *a[2]	6. 8
7. (*a)[2]	7. 2
8. *(*a + 2)	8. 2

解析：

记 int [4]类型为 T，那么 a 是 T [3]类型的数组，d 是指向 T 类型的指针

d = a + 1 意味着 d 指向 a 数组第 1 个元素。

例题

8. 在 x86-64 架构下，有如下变量：

```
union {char c[8], int i;} x;
```

在 x.i=0x41424344 时，x.c[2]的值为多少（提示：'A'=0x41）：

A. 'A'　　B. 'B'　　C. 'C'　　D. 'D'

例题

8. 在 x86-64 架构下，有如下变量：

```
union {char c[8], int i;} x;
```

在 x.i=0x41424344 时，x.c[2]的值为多少（提示：'A'=0x41）：

A. 'A'　　B. 'B'　　C. 'C'　　D. 'D'

B

小端法

缓冲区溢出攻击

- 覆盖返回地址
  - 防御方式:
    - 栈随机化 / 基址随机化 (PIE) ⊆ ASLR（地址空间随机化）

      - 对抗: nop sled (+暴力尝试)
    - Canary Value 金丝雀值 **%fs:0x28**

      - 对抗: 尝试复制/输出canary value; 暴力尝试
    - 将非代码段标记为NX (non-executed, 不可执行)

      - 对抗: ROP (return-oriented programming), 利用原有代码中的片段

RISC / CISC

- R = reduced, C = complex
- RISC vs. CISC (选择题会考)

CISC

- **Many** types of **complex** instructions
- Some with **long** execution times
  - **Resemble higher level language**
- **Variable-size** encodings
- Multiple addressing formats

Early RISC

- **Fewer** types of instructions
- **Short** execution times
- **Fixed-length** encodings
- **Simple** addressing formats: **base and displacement addressing** only

RISC / CISC

- R = reduced, C = complex
- RISC vs. CISC（选择题会考)

CISC

- Emphasis on **hardware**
- Memory-to-memory
- **Small** code size
- **High** cycles per instruction

Early RISC

- Emphasis on **software**
- Register-to-register
  - Load-store architecture
- **Large** code size
- **Low** cycles per instruction

RISC / CISC

CISC	早期的 RISC
指令数量很多。Intel 描述全套指令的文档[51]有 1200 多页。	指令数量少得多。通常少于 100 个。
有些指令的延迟很长。包括将一个整块从内存的一个部分复制到另一部分的指令，以及其他一些将多个寄存器的值复制到内存或从内存复制到多个寄存器的指令。	没有较长延迟的指令。有些早期的 RISC 机器甚至没有整数乘法指令，要求编译器通过一系列加法来实现乘法。
编码是可变长度的。x86-64 的指令长度可以是 1～15 个字节。	编码是固定长度的。通常所有的指令都编码为 4 个字节。
指定操作数的方式很多样。在 x86-64 中，内存操作数指示符可以有许多不同的组合，这些组合由偏移量、基址和变址寄存器以及伸缩因子组成。	简单寻址方式。通常只有基址和偏移量寻址。
可以对内存和寄存器操作数进行算术和逻辑运算。	只能对寄存器操作数进行算术和逻辑运算。允许使用内存引用的只有 load 和 store 指令，load 是从内存读到寄存器，store 是从寄存器写到内存。这种方法被称为 load/store 体系结构。
对机器级程序来说实现细节是不可见的。ISA 提供了程序和如何执行程序之间的清晰的抽象。	对机器级程序来说实现细节是可见的。有些 RISC 机器禁止某些特殊的指令序列，而有些跳转要到下一条指令执行完了以后才会生效。编译器必须在这些约束条件下进行性能优化。
有条件码。作为指令执行的副产品，设置了一些特殊的标志位，可以用于条件分支检测。	没有条件码。相反，对条件检测来说，要用明确的测试指令，这些指令会将测试结果放在一个普通的寄存器中。
栈密集的过程链接。栈被用来存取过程参数和返回地址。	寄存器密集的过程链接。寄存器被用来存取过程参数和返回地址。因此有些过程能完全避免内存引用。通常处理器有更多的(最多的有 32 个)寄存器。

例

(　　)1. 下面有三组对于指令集的描述，它们分别符合①____，②____，③____ 的特点。

① 某指令集中，只有两条指令能够访问内存。

② 某指令集中，指令的长度都是4字节。

③ 某指令集中，可以只利用一条指令完成字符串的复制，也可以只利用一条指令查找字符串中第一次出现字母K的位置。

A. CISC, CISC, CISC

B. RISC, RISC, CISC

C. RISC, CISC, RISC

D. CISC, RISC, RISC

例

(　　)1. 下面有三组对于指令集的描述，它们分别符合①____，②____，③____ 的特点。
① 某指令集中，只有两条指令能够访问内存。
② 某指令集中，指令的长度都是4字节。
③ 某指令集中，可以只利用一条指令完成字符串的复制，也可以只利用一条指令查找字符串中第一次出现字母K的位置。
A. CISC, CISC, CISC
B. RISC, RISC, CISC
C. RISC, CISC, RISC
D. CISC, RISC, RISC

B

例

指令机器码长度固定
指令类型多、功能丰富
不采用条件码
实现同一功能，需要的汇编代码较多
译码电路复杂
访存模式多样
参数、返回地址都使用寄存器进行保存
x86-64
MIPS
广泛用于嵌入式系统
已知某个体系结构使用 add R1,R2,R3 来完成加法运算。当要将数据从寄存器 S 移动至寄存器 D 时，使用 add S,#ZR,D 进行操作（#ZR 是一个恒为 0 的寄存器），而没有类似于 mov 的指令。
已知某个体系结构提供了 xlat 指令，它以一个固定的寄存器 A 为基地址，以另一个固定的寄存器 B 为偏移量，在 A 对应的数组中取出下标为 B 的项的内容，放回寄存器 A 中。

例

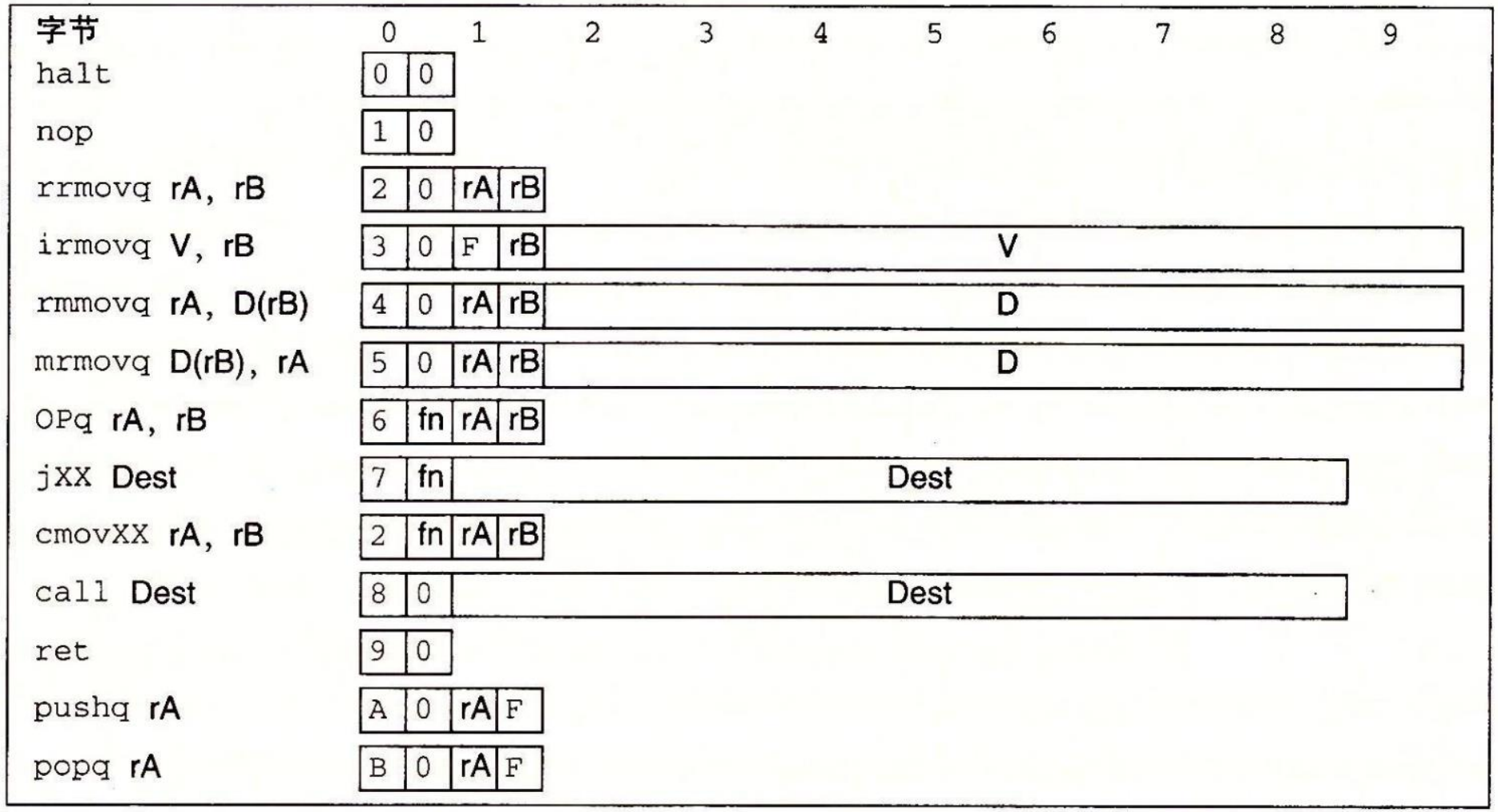
指令机器码长度固定
指令类型多、功能丰富
不采用条件码
实现同一功能，需要的汇编代码较多
译码电路复杂
访存模式多样
参数、返回地址都使用寄存器进行保存
x86-64
MIPS
广泛用于嵌入式系统
已知某个体系结构使用 add R1,R2,R3 来完成加法运算。当要将数据从寄存器 S 移动至寄存器 D 时，使用 add S,#ZR,D 进行操作（#ZR 是一个恒为 0 的寄存器），而没有类似于 mov 的指令。
已知某个体系结构提供了 xlat 指令，它以一个固定的寄存器 A 为基地址，以另一个固定的寄存器 B 为偏移量，在 A 对应的数组中取出下标为 B 的项的内容，放回寄存器 A 中。

R C R R C

C R C R R(ARM等)

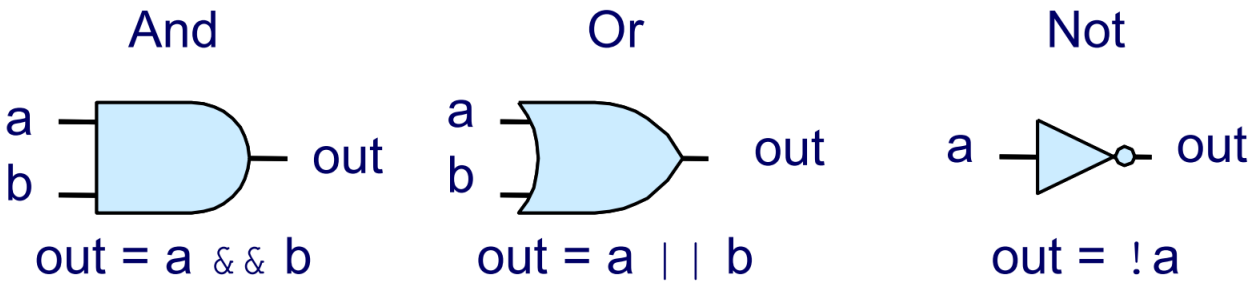
R C

Y86-64


字节
0 1 2 3 4 5 6 7 8 9
halt 0 0
nop 1 0
rrmovq rA, rB 2 0 rA rB
irmovq V, rB 3 0 F rB V
rmmovq rA, D(rB) 4 0 rA rB D
mrmovq D(rB), rA 5 0 rA rB D
OPq rA, rB 6 fn rA rB
jXX Dest 7 fn Dest
cmovXX rA, rB 2 fn rA rB
call Dest 8 0 Dest
ret 9 0
pushq rA A 0 rA F
popq rA B 0 rA F

Y86-64

- 实验性质的汇编语言, 但必须熟练掌握 (考试和lab都要用)
- 工具链:
  - YAS: Y86-64 assembler (.ys → .yo)
  - YIS: Y86-64 simulator (run .yo programs)

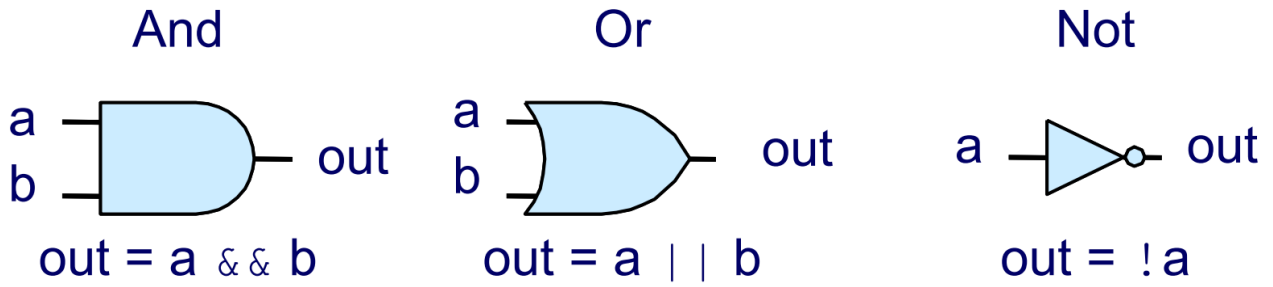


值	名字	含义
1	AOK	正常操作
2	HLT	遇到器执行 halt 指令
3	ADR	遇到非法地址
4	INS	遇到非法指令

图 4-5 Y86-64 状态码。在我们的设计中，任何 AOK 以外的代码都会使处理器停止

逻辑电路和HCL

- 逻辑门



- 多路复用器

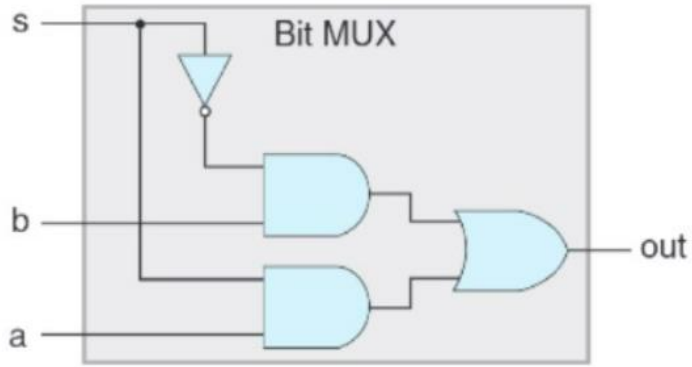


Figure 4.11 **Single-bit multiplexor circuit.** The output will equal input a if the control signal s is 1 and will equal input b when s is 0.

- ALU = Arithmetic/Logic Unit

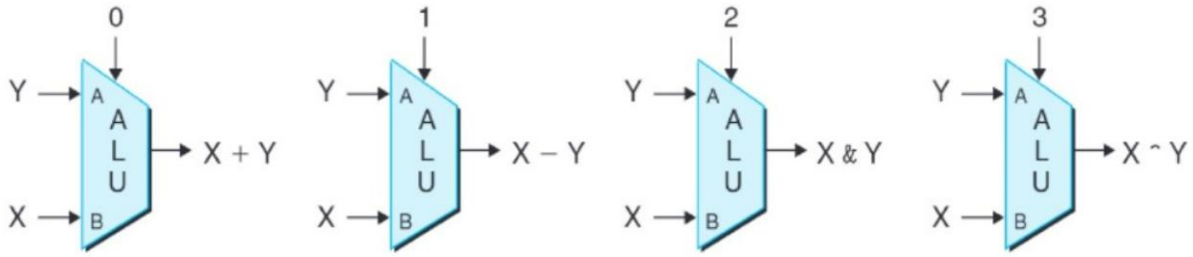
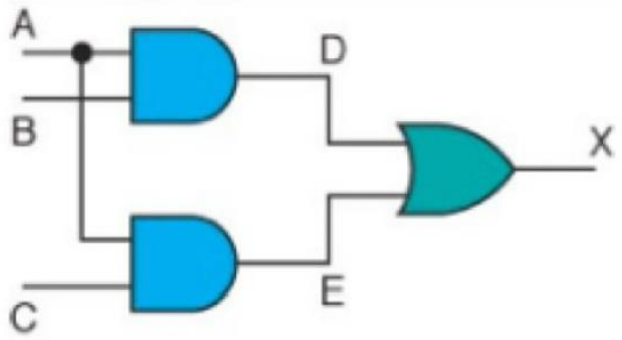


Figure 4.15 **Arithmetic/logic unit (ALU).** Depending on the setting of the function input, the circuit will perform one of four different arithmetic and logical operations.

例

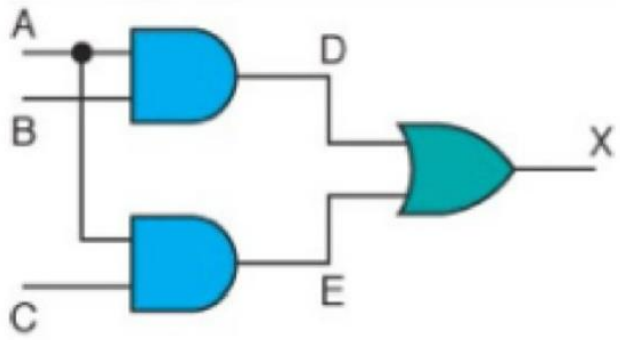
(　　)4. 对应下述组合电路的正确 HCL 表达式为



A. Bool X = (A || B) && (A || C)
B. Bool X = A || (B && C)
C. Bool X = A && (B || C)
D. Bool X = A || B || C

例

(　　)4. 对应下述组合电路的正确 HCL 表达式为



A. Bool X = (A || B) && (A || C)
B. Bool X = A || (B && C)
C. Bool X = A && (B || C)
D. Bool X = A || B || C

C

(A && B) || (A && C) = A && (B || C)

Thank you!