

C 语言拾遗(2): 编程实践

王慧妍

why@nju.edu.cn

南京大学



计算机科学与技术系



计算机软件研究所



本讲概述

PA1已悄悄发布：（OJ暂未开启PA1评测）

* PA 1.1: 2023.9.30 (此为建议的不计分 deadline)

* PA 1.2: 2023.10.5 (此为建议的不计分 deadline)

* PA 1.3: 2023.10.15 23:59:59 (以此 deadline 计按时提交bonus)

- 假设你已经熟练使用C语言的各种机制（并没有）
 - 原则上给需求就能搞定任何代码（并不是）
- 本次课程
 - 怎样写代码才能从一个大型项目中存活下来？
 - 核心准则：编写可读代码
 - 两个例子：计数器，YEMU

核心准则：编写可读代码

一个极端不可读的例子

- [IOCCC'11 best self documenting program](#)
 - 不可读 = 不可维护

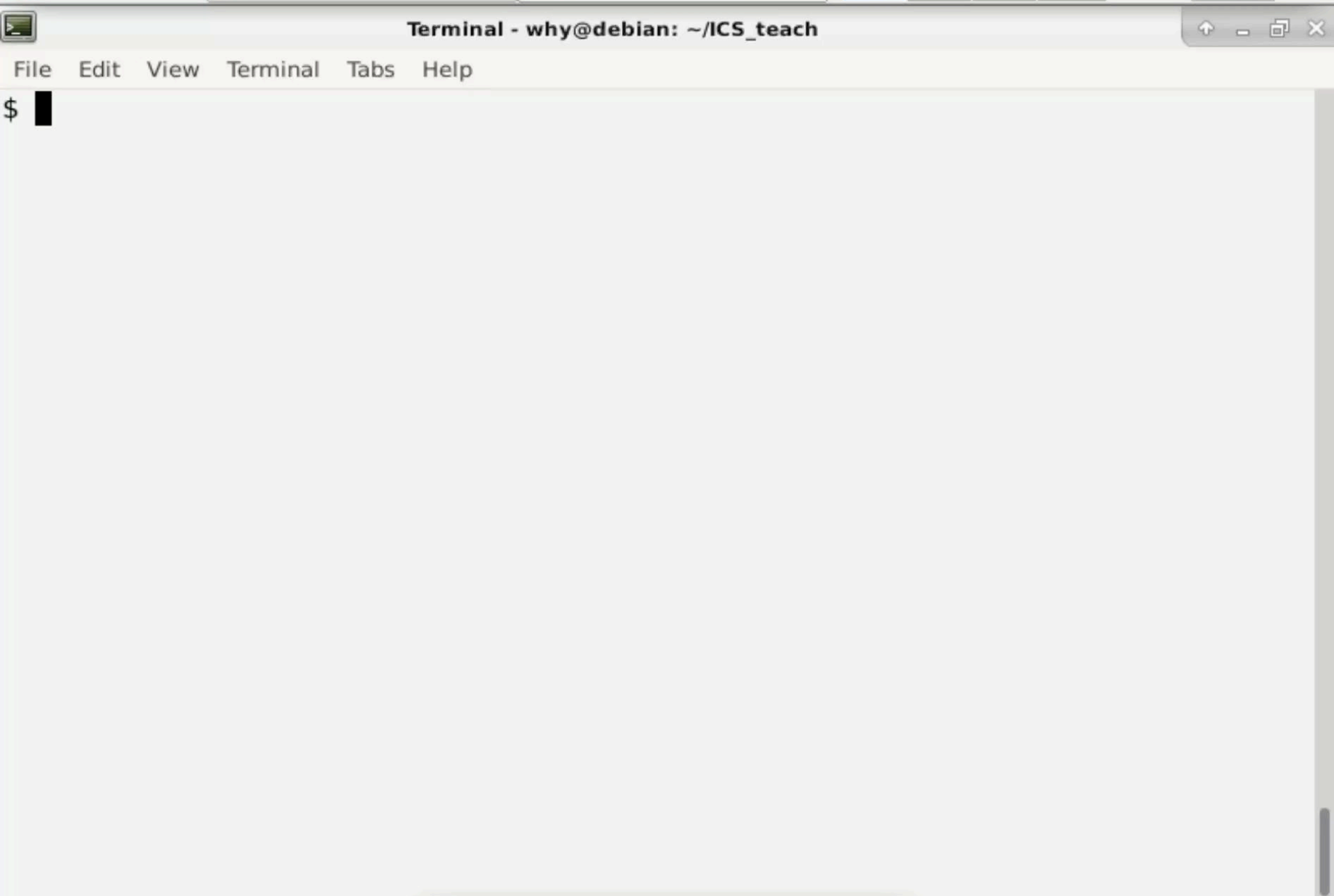
```
puts(usage: calculator 11/26+222/31
+~~~~~calculator-\
!
!              7.584,367 )
+~~~~~+
! clear ! 0 ||1 -x 1 tan I (/) |
+~~~~~+
! 1 | 2 | 3 ||1 1/x 1 cos I (*) |
+~~~~~+
! 4 | 5 | 6 ||1 exp 1 sqrt I (+) |
+~~~~~+
! 7 | 8 | 9 ||1 sin 1 log I (-) |
+~~~~~(0 )
```

一个极端不可读的例子

- IOCCC'11 best self documenting program

- 不可读 = 不可维护

```
#define clear 1;
    if(c>=11){c=0;sscanf(_,"%lf%c",&r,&c);while(*++_-
c);} \    else if(argc>=4&&!main(4-
(*_++=='('),argv))_++;g:c+=
#define puts(d,e) return 0;}{double a;int b;char
    c=(argc<4?d)&15;\ b=(*_%__LINE__+7)%9*(3*e>>c&1);c+=
#define I(d)
(r);if(argc<4&&*#d==*_){a=r;r=usage?r*a:r+a;goto
    g;}c=c
#define return if(argc==2)printf("%f\n",r);return
argc>=4+ #define usage main(4-__LINE__/26,argv)
#define calculator *_*(int)
#define l (r);r=--b?r:
#define _ argv[1]
```



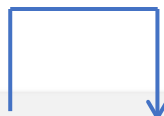
一个现实中可能遇到的例子

- 人类不可读版本 (STFW: clockwise/spiral rule)
 - 终极使用顺时针螺旋法则的案例

```
void (*signal (int sig, void (*func)(int)))(int);
```

一个现实中可能遇到的例子

- 人类不可读版本 (STFW: clockwise/spiral rule)
 - 终极使用顺时针螺旋法则的案例

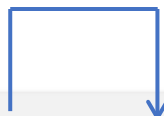


```
void (*signal (int sig, void (*func)(int)))(int);
```

- signal是一个函数
 - 参数为.....
 - 返回值为.....

一个现实中可能遇到的例子

- 人类不可读版本 (STFW: clockwise/spiral rule)
 - 终极使用顺时针螺旋法则的案例



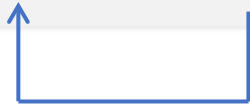
```
void (*signal (int sig, void (*func)(int)))(int);
```

- signal是一个函数
 - 参数为int和一个函数指针 (参数为int, 返回值为void)
 - 返回值为.....

一个现实中可能遇到的例子

- 人类不可读版本 (STFW: clockwise/spiral rule)
 - 终极使用顺时针螺旋法则的案例

```
void (*signal (int sig, void (*func)(int)))(int);
```



- signal是一个函数
 - 参数为int和一个函数指针 (参数为int, 返回值为void)
 - 返回值为一个.....指针

一个现实中可能遇到的例子

- 人类不可读版本 (STFW: clockwise/spiral rule)
 - 终极使用顺时针螺旋法则的案例



```
void (*signal (int sig, void (*func)(int))) (int);
```

- signal是一个函数
 - 参数为int和一个指向函数的指针 (函数参数为int, 返回值为void)
 - 返回值为一个指向函数的指针 (参数为..., 返回值为...)

一个现实中可能遇到的例子

- 人类不可读版本 (STFW: clockwise/spiral rule)
 - 终极使用顺时针螺旋法则的案例



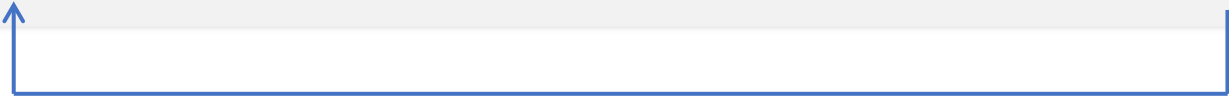
```
void (*signal (int sig, void (*func)(int)))(int);
```

- signal是一个函数
 - 参数为int和一个指向函数的指针 (函数参数为int, 返回值为void)
 - 返回值为一个指向函数的指针 (参数为int, 返回值为...)

一个现实中可能遇到的例子

- 人类不可读版本 (STFW: clockwise/spiral rule)
 - 终极使用顺时针螺旋法则的案例

```
void (*signal (int sig, void (*func)(int)))(int);
```



- signal是一个函数
 - 参数为int和一个指向函数的指针 (函数参数为int, 返回值为void)
 - 返回值为一个指向函数的指针 (参数为int, 返回值为void)
- 太复杂了>_<, 有没有更简单的办法

一个现实中可能遇到的例子

- 人类不可读版本 (STFW: clockwise/spiral rule)
 - 终极使用顺时针螺旋法则

```
void (*signal (int sig, void (*func)(int)))(int);
```

- signal是一个函数
 - 参数为int和一个指向函数的指针 (函数参数为int, 返回值为void)
 - 返回值为一个指向函数的指针 (参数为int, 返回值为void)

```
void (*signal (int sig, void (*func)(int)))(int);
```

```
void (*signal (int sig, func))(int);
```

```
void (*)(int);
```

一个现实中可能遇到的例子

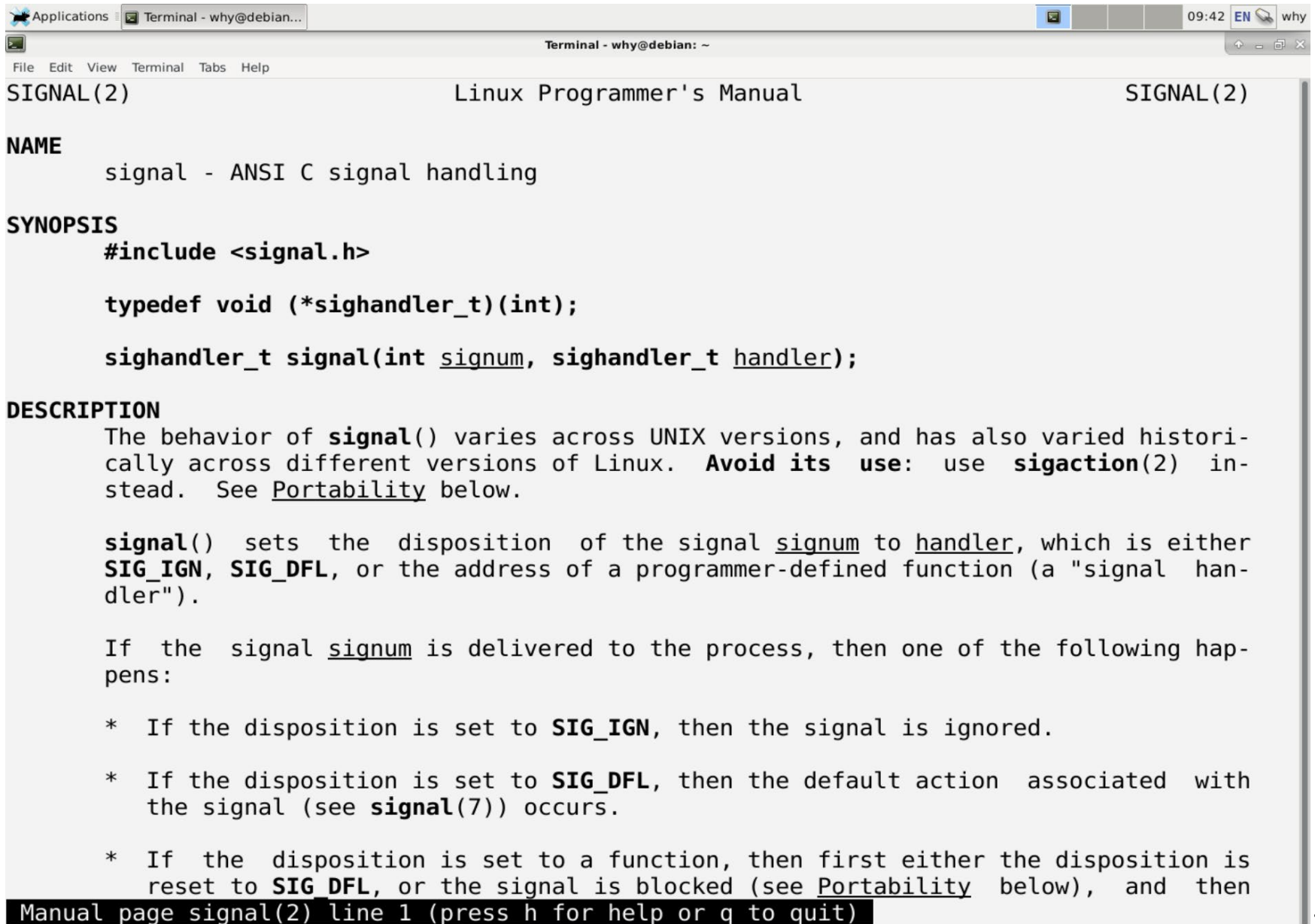
- 人类不可读版本 (STFW: clockwise/spiral rule)
 - 终极使用顺时针螺旋法则

```
void (*signal (int sig, void (*func)(int)))(int);
```

- 人类可读版本

```
typedef void (*sighandler_t)(int);  
sighandler_t signal(int, sighandler_t);
```

man 2 signal



```
Applications Terminal - why@debian... 09:42 EN why
Terminal - why@debian: ~
File Edit View Terminal Tabs Help
SIGNAL(2) Linux Programmer's Manual SIGNAL(2)

NAME
    signal - ANSI C signal handling

SYNOPSIS
    #include <signal.h>

    typedef void (*sighandler_t)(int);

    sighandler_t signal(int signum, sighandler_t handler);

DESCRIPTION
    The behavior of signal() varies across UNIX versions, and has also varied historically across different versions of Linux. Avoid its use: use sigaction(2) instead. See Portability below.

    signal() sets the disposition of the signal signum to handler, which is either SIG_IGN, SIG_DFL, or the address of a programmer-defined function (a "signal handler").

    If the signal signum is delivered to the process, then one of the following happens:

    * If the disposition is set to SIG_IGN, then the signal is ignored.

    * If the disposition is set to SIG_DFL, then the default action associated with the signal (see signal(7)) occurs.

    * If the disposition is set to a function, then first either the disposition is reset to SIG_DFL, or the signal is blocked (see Portability below), and then
```

Manual page signal(2) line 1 (press h for help or q to quit)

编写代码的准则：降低维护成本

Programs are meant to be read by humans and only incidentally for computers to execute. — D. E. Knuth

(程序首先是拿给人读的，其次才是被机器执行。)

- 宏观

- 做好分解和解耦（现实世界也是这样管理复杂系统）

- 微观

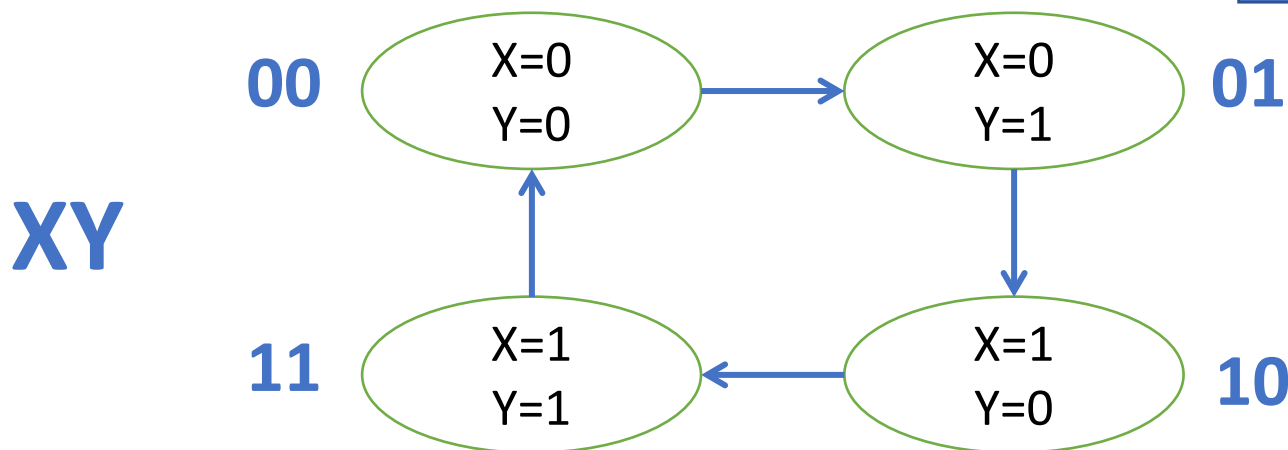
- “不言自明”
 - 通过阅读代码能够理解一段程序的行为 (implementation)
- “不言自证”
 - 通过阅读代码能够验证一段程序implementation与specification的一致性

例子：实现数字逻辑电路模拟器

数字逻辑电路模拟器

- 假想的数字逻辑电路

- 若干个1-bit边沿触发寄存器 (X, Y,)
- 若干个逻辑门



- 基本思路：状态（存储模拟）+计算模拟

- 状态 = 变量
 - int X = 0, Y = 0;
- 计算

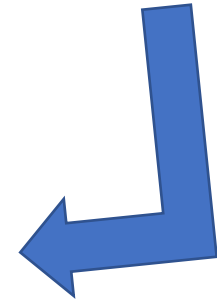
```
X1 = (!X && Y) || (X && !Y);
Y1 = !Y;
X = X1; Y = Y1;
```

数字逻辑电路模拟器

- 需求
 - 加一位边沿寄存器
 - 自己独立的逻辑

```
int X=0, Y=0;
int X1=0, Y1=0;
while(1){
    X1 = (!X&&Y) || (X&&!Y);
    Y1 = !Y;
    X = X1; Y = Y1;
}
```

```
int X=0, Y=0, Z=0;
int X1=0, Y1=0, Z1=0;
while(1){
    X1 = (!X&&Y) || (X&&!Y);
    Y1 = !Y;
    Z1 = .....;
    X = X1; Y = Y1; Z = Z1;
}
```



通用数字逻辑电路模拟器

```
#define FORALL_REGS(_)    _(X) _(Y)
#define LOGIC             X1 = (!X&&Y)|| (X&&!Y); \
                          Y1 = !Y;

#define DEFINE(X)         static int X, X##1;
#define UPDATE(X)         X = X##1;
#define PRINT(X)          printf("#X " = %d; ", X);
int main() {
    FORALL_REGS(DEFINE);

    while (1) { // clock
        FORALL_REGS(PRINT);
        putchar('\n');
        sleep(1);
        LOGIC;
        FORALL_REGS(UPDATE);
    }
}
```

Terminal - a.c (~/ICS_teach) - VIM

File Edit View Terminal Tabs Help

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #define FORALL_REGS(_) _ (X) _ (Y)
4 #define LOGIC          X1 = (!X && Y) || (X&&!Y); \
5                        Y1 = !Y;
6 #define DEFINE(X)      static int X, X##1;
7 #define UPDATE(X)      X = X##1;
8 #define PRINT(X)       printf("#X " = %d; ", X);
9
10 int main() {
11     FORALL_REGS(DEFINE);
12     while (1) { // clock
13         FORALL_REGS(PRINT); putchar('\n'); sleep(1);
14         LOGIC;
15         FORALL_REGS(UPDATE);
16     }
17 }
18
```

a.c 3,1 All

"a.c" 18L, 437C

Applications C 语言拾遗 (2): 编程实... Terminal - a.c (~/ICS_... 11:54 EN why

Terminal - a.c (~/ICS_teach) - VIM

File Edit View Terminal Tabs Help

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #define FORALL_REGS(_) _ (X) _ (Y) _ (Z)
4 #define LOGIC          X1 = (!X&&Y&&Z) || (X&&(!(Y&&Z))); \
5                        Y1 = (!Y&&Z) || (Y&&!Z); \
6                        Z1 = !Z;
7 #define DEFINE(X)      static int X, X##1;
8 #define UPDATE(X)      X = X##1;
9 #define PRINT(X)       printf(#X " = %d; ", X);
10
11 int main() {
12     FORALL_REGS(DEFINE);
13     while (1) { // clock
14         FORALL_REGS(PRINT); putchar('\n'); sleep(1);
15         LOGIC;
16         FORALL_REGS(UPDATE);
17     }
18 }
19
```

a.c 4,1 All
"a.c" 19L, 498C

通用数字逻辑电路模拟器

```
#define FORALL_REGS(_)    _(X) _(Y)
#define LOGIC             X1 = (!X&&Y)|| (X&&!Y); \
                          Y1 = !Y;

#define DEFINE(X)         static int X, X##1;
#define UPDATE(X)         X = X##1;
#define PRINT(X)          printf("#X " = %d; ", X);
int main() {
    FORALL_REGS(DEFINE);

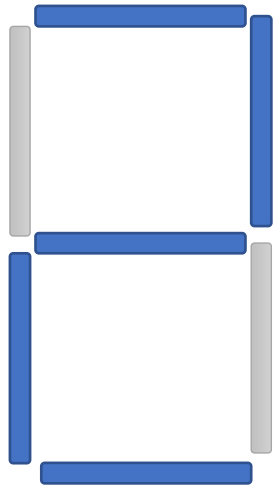
    while (1) { // clock
        FORALL_REGS(PRINT);
        putchar('\n');
        sleep(1);
        LOGIC;
        FORALL_REGS(UPDATE);
    }
}
```


使用预编译：Pros and Cons

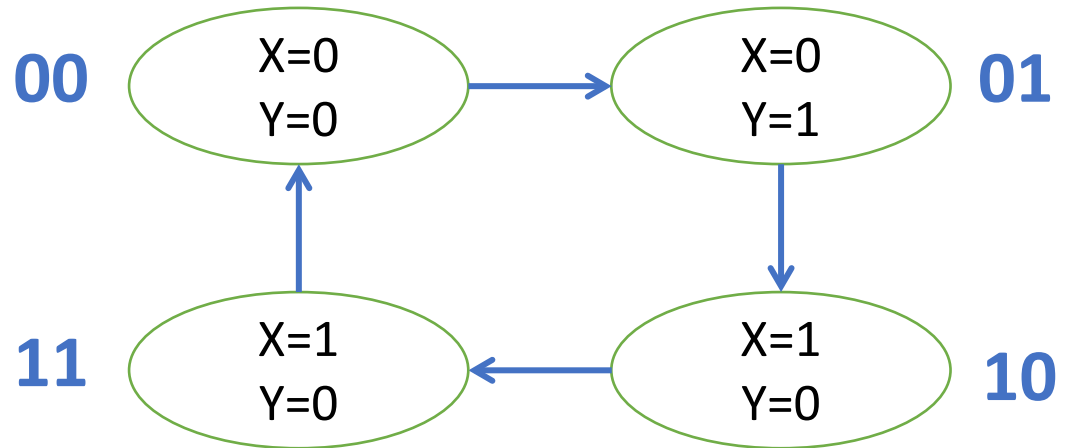
- Pros
 - 增加/删除寄存器只要改很少的地方
 - 阻止一些编程错误
 - 忘记更新寄存器
 - 忘记打印寄存器
 - “不言自明”
- Cons
 - 可读性变差（不太像C代码）
 - “不言自证” 差一些
 - 给IDE解析带来一些困难

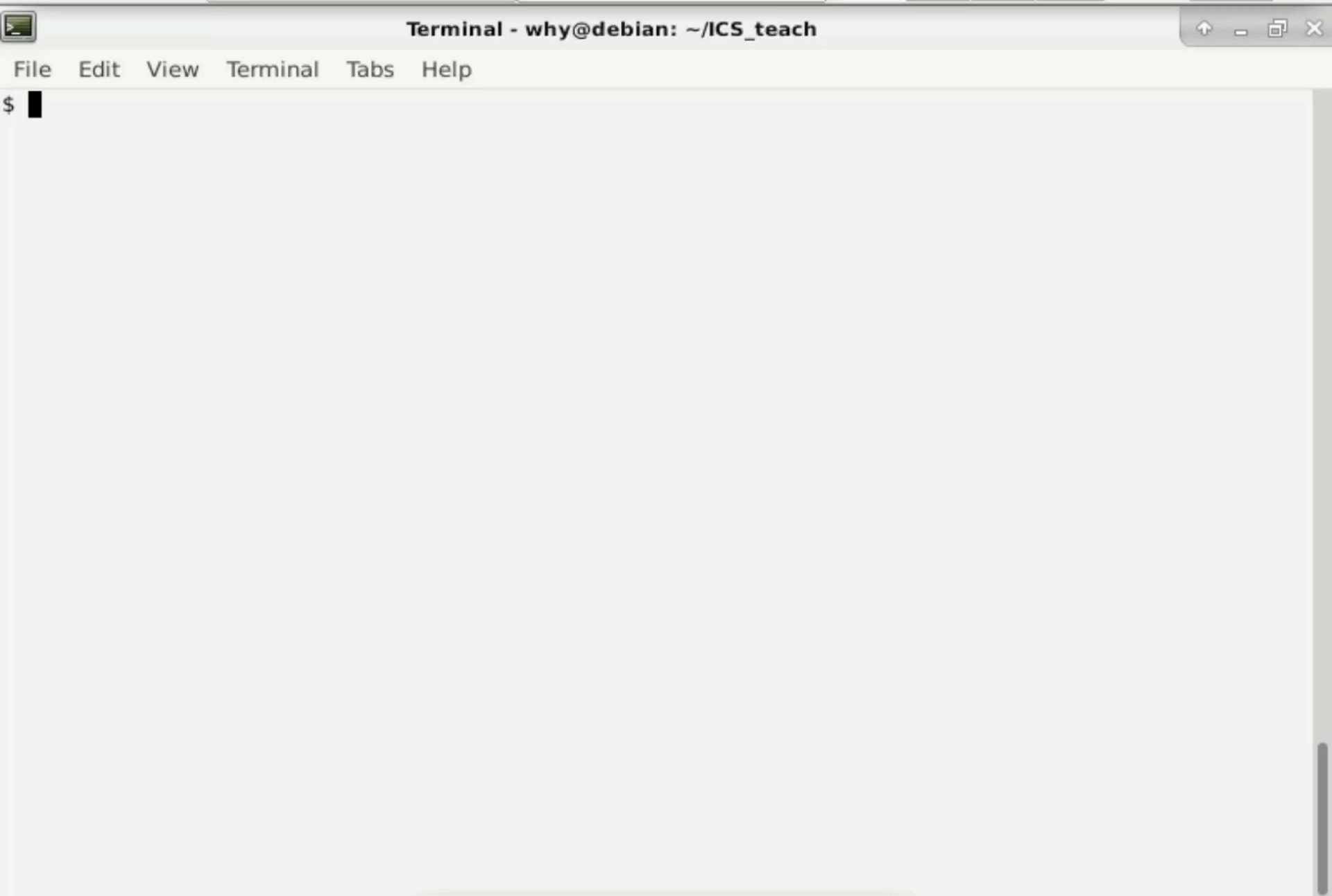
更完整的实现：数码管显示

- logisim.c 和 display.py



XY





```

#include <stdio.h>
#include <unistd.h>
#define FORALL_REGS(_)      _(X) _(Y)
#define OUTPUTS(_)         _(A) _(B) _(C) _(D) _(E) _(F) _(G)
#define LOGIC
                            X1 = (!X && Y) || (X && !Y); \
                            Y1 = !Y; \
                            A  = (!X && !Y) || (X && !Y) || (X && Y); \
                            B  = 1; \
                            C  = (!X && !Y) || (!X && Y) || (X && Y); \
                            D  = (!X && !Y) || (X && !Y) || (X && Y); \
                            E  = (!X && !Y) || (X && !Y); \
                            F  = (!X && !Y); \
                            G  = (X && !Y) || (X && Y);
#define DEFINE(X)           static int X, X##1;
#define UPDATE(X)          X = X##1;
#define PRINT(X)            printf(#X " = %d; ", X);
int main() {
    FORALL_REGS(DEFINE);
    OUTPUTS(DEFINE);
    while (1) { // clock
        LOGIC;
        OUTPUTS(PRINT);
        putchar('\n');
        fflush(stdout);
        sleep(1);
        FORALL_REGS(UPDATE);
    }
}

```

更完整的实现：数码管显示

- logisim.c 和 display.py
 - 也可以考虑增加更多外设：开关、UART等
 - 原理无限接近大家数字电路课玩过的FPGA
- 等等.....FPGA?
 - 这玩意不是万能的吗？？？
 - 我们能模拟它，是不是就能模拟计算机系统？
 - Yes!
 - 我们实现了一个超级超级低配版 NEMU!



例子：实现YEMU全系统模拟器

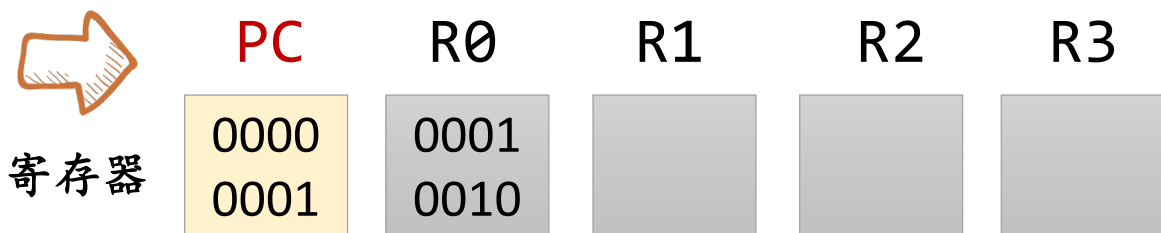
教科书第一章上的 “计算机系统”

• 存储系统and指令集

寄存器: PC, R0 (RA), R1, R2, R3
(8-bit)

内存: 16字节 (按字节访问)

	7	6	5	4	3	2	1	0
mov	[0	0	0	0]	[	rt]	[	rs]
add	[0	0	0	1]	[	rt]	[	rs]
load	[1	1	1	0]	[	addr		]
store	[1	1	1	1]	[	addr		]



内存

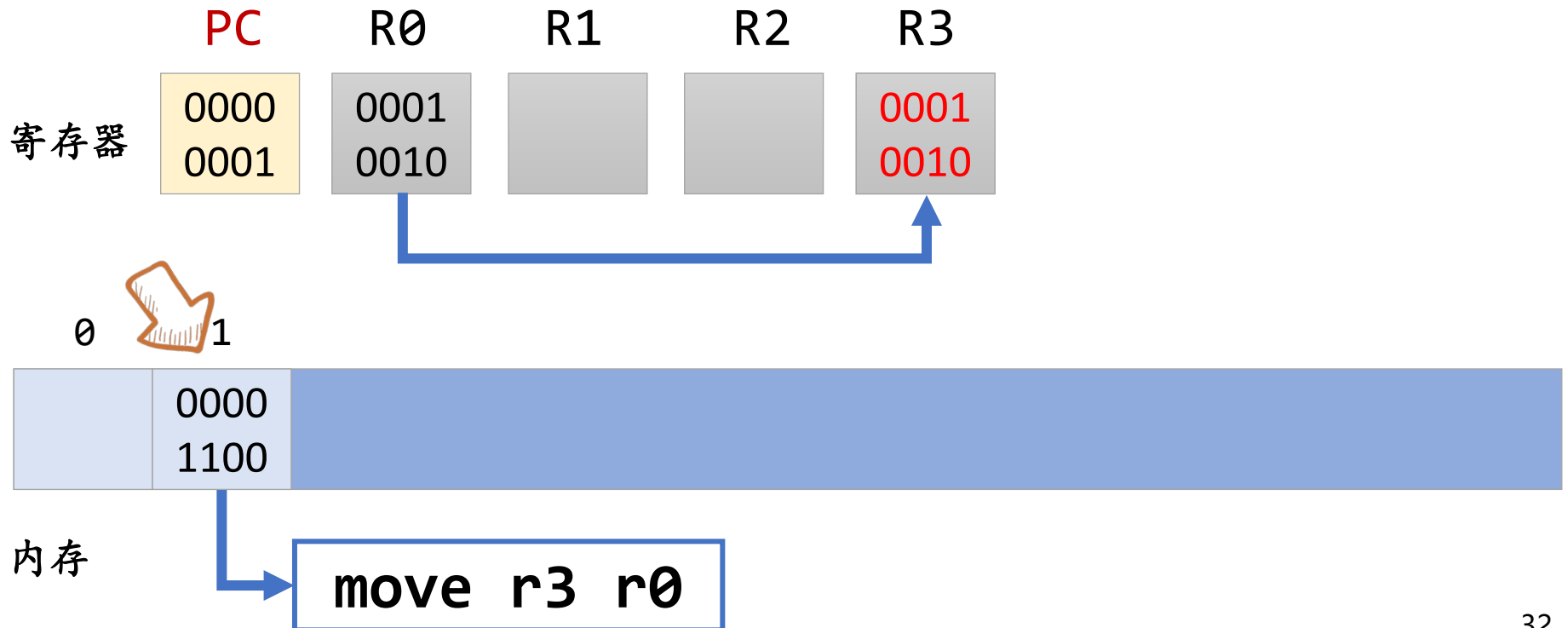
教科书第一章上的“计算机系统”

• 存储系统and指令集

寄存器: PC, R0 (RA), R1, R2, R3
(8-bit)

内存: 16字节 (按字节访问)

	7	6	5	4	3	2	1	0
mov	[0	0	0	0]	[	rt]	[	rs]
add	[0	0	0	1]	[	rt]	[	rs]
load	[1	1	1	0]	[	addr		
store	[1	1	1	1]	[	addr		



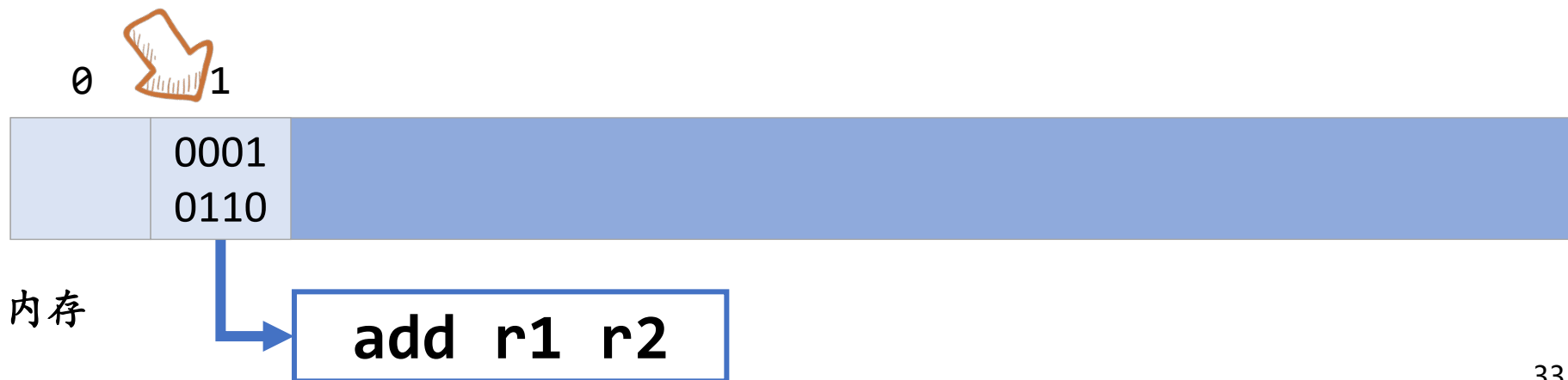
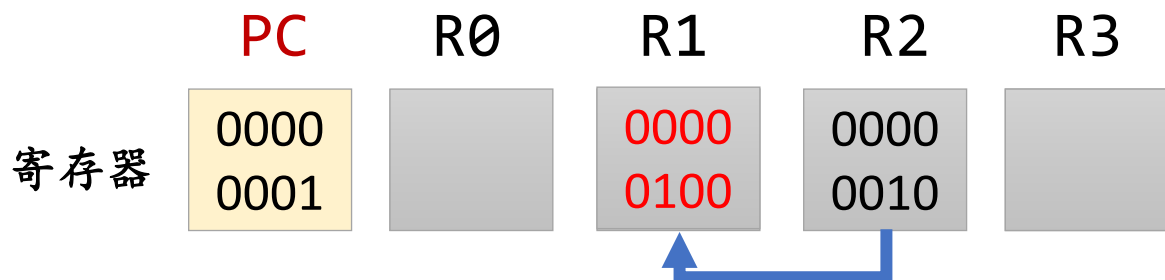
教科书第一章上的 “计算机系统”

• 存储系统and指令集

寄存器: PC, R0 (RA), R1, R2, R3
(8-bit)

内存: 16字节 (按字节访问)

	7	6	5	4	3	2	1	0
mov	[0	0	0	0]	[	rt]	[	rs]
add	[0	0	0	1]	[	rt]	[	rs]
load	[1	1	1	0]	[	addr		]
store	[1	1	1	1]	[	addr		]



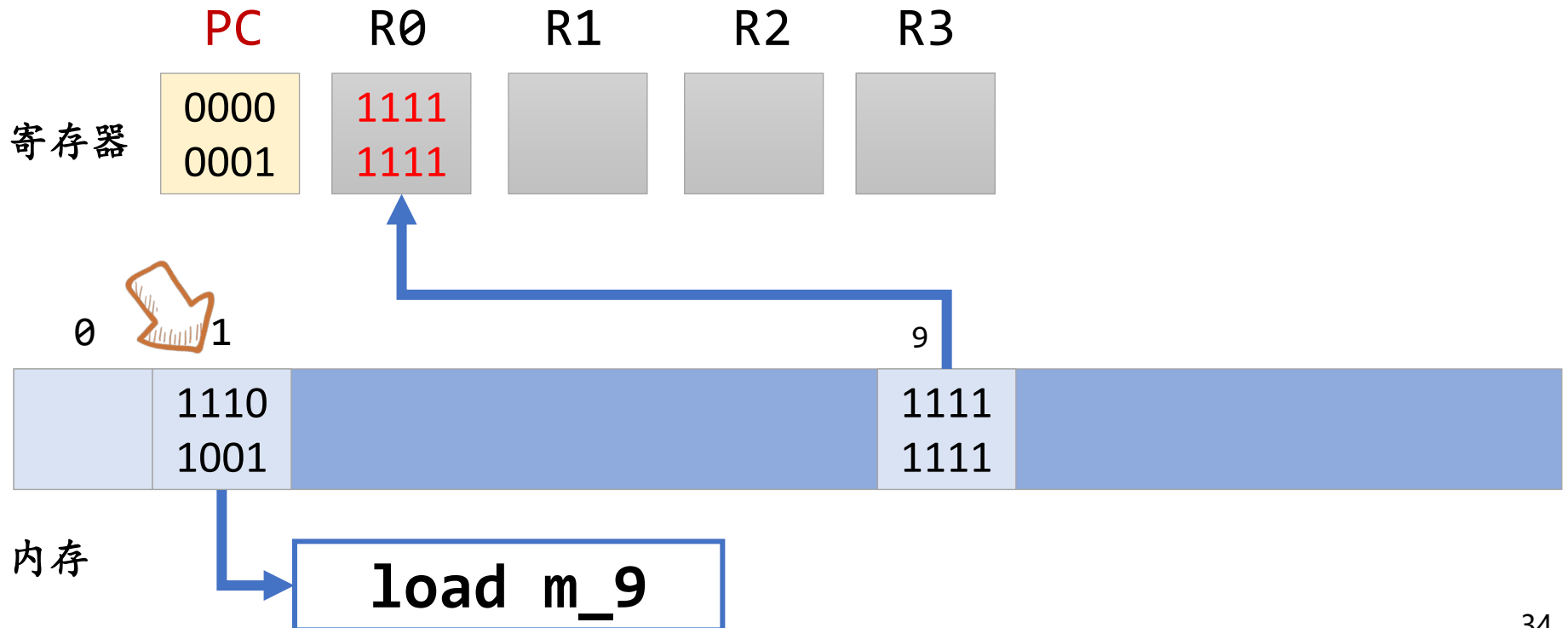
教科书第一章上的“计算机系统”

• 存储系统and指令集

寄存器: PC, R0 (RA), R1, R2, R3
(8-bit)

内存: 16字节 (按字节访问)

	7	6	5	4	3	2	1	0
mov	[0	0	0	0]	[	rt]	[	rs]
add	[0	0	0	1]	[	rt]	[	rs]
load	[1	1	1	0]	[	addr		]
store	[1	1	1	1]	[	addr		]



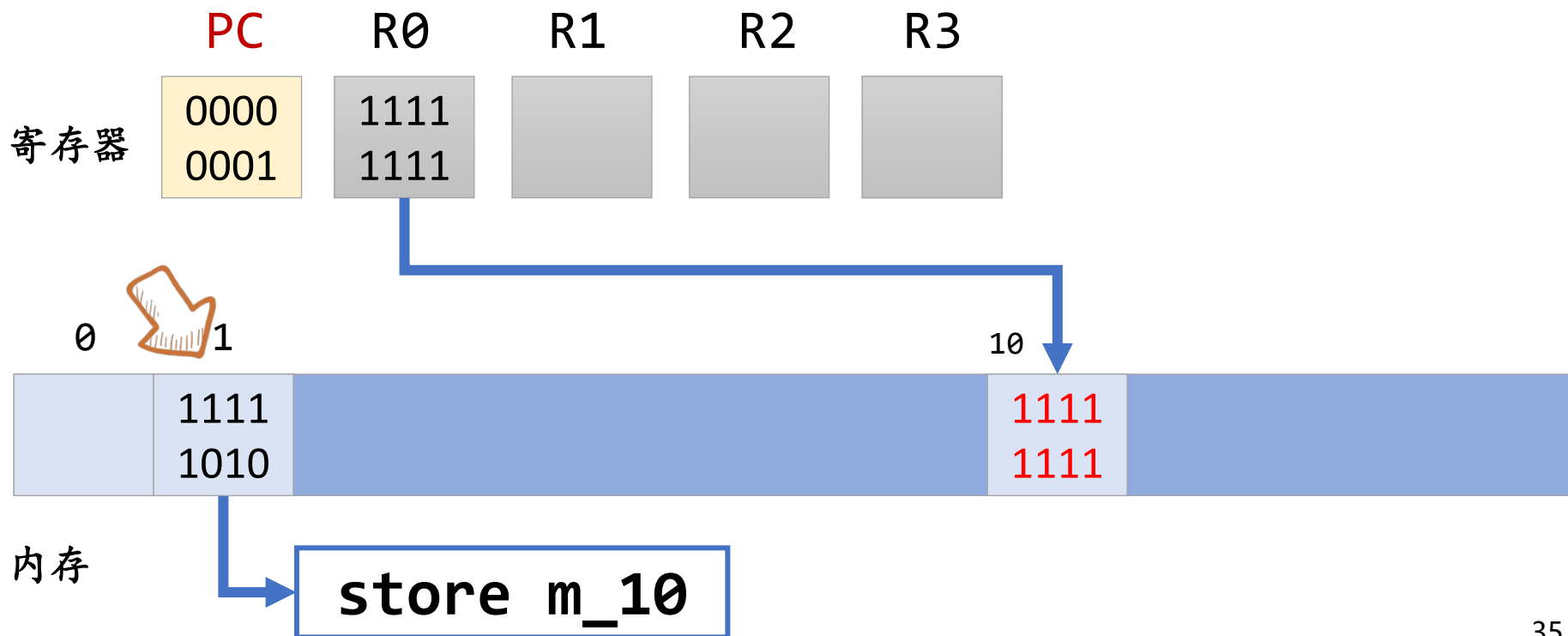
教科书第一章上的 “计算机系统”

• 存储系统and指令集

寄存器: PC, R0 (RA), R1, R2, R3
(8-bit)

内存: 16字节 (按字节访问)

	7	6	5	4	3	2	1	0
mov	[0	0	0	0]	[	rt]	[	rs]
add	[0	0	0	1]	[	rt]	[	rs]
load	[1	1	1	0]	[	addr		]
store	[1	1	1	1]	[	addr		]



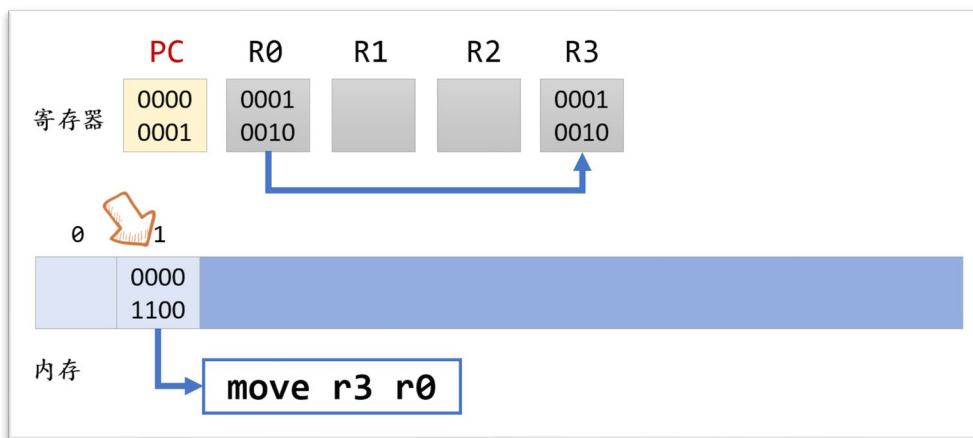
教科书第一章上的“计算机系统”

- 存储系统and指令集

寄存器：PC，R0 (RA)，R1，R2，R3
(8-bit)

内存：16字节（按字节访问）

	7	6	5	4	3	2	1	0
mov	[0	0	0	0]	[rt]	[rs]		
add	[0	0	0	1]	[rt]	[rs]		
load	[1	1	1	0]	[addr			
store	[1	1	1	1]	[addr			



- 有“计算机系统”的感觉了？

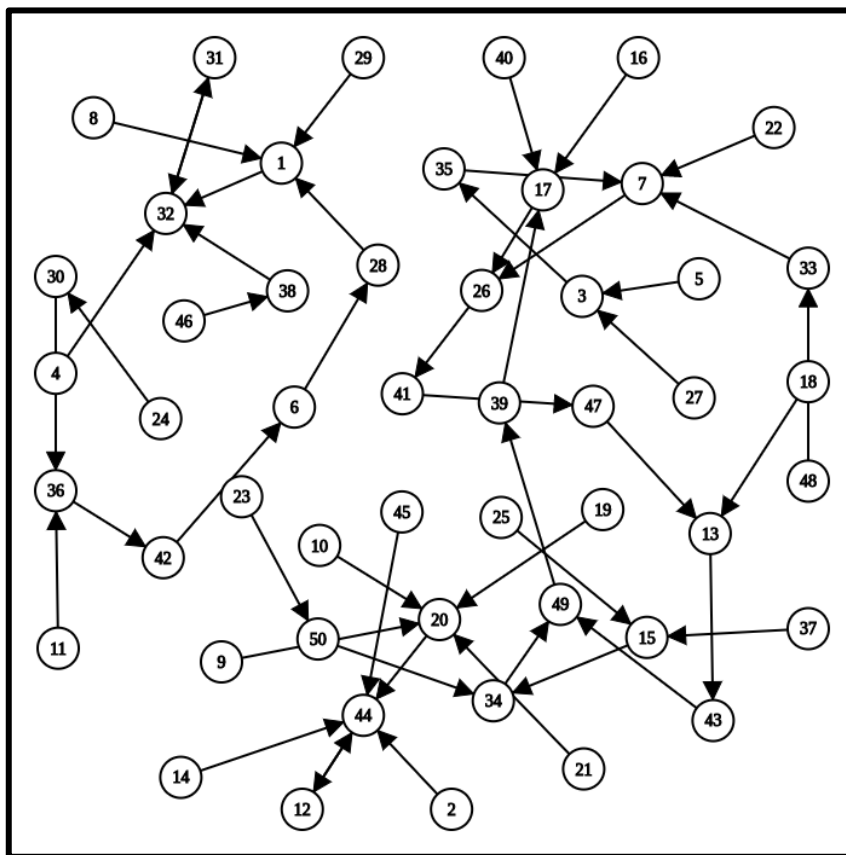
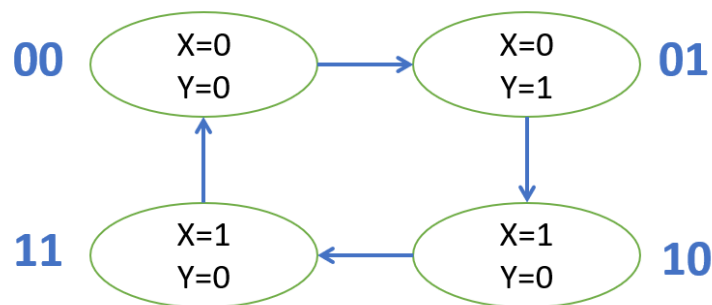
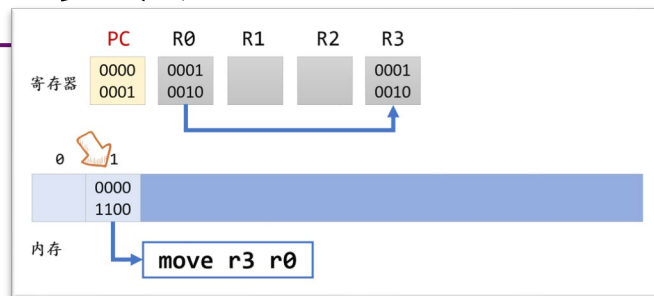
- 它显然可以用数字逻辑电路实现
- 不过我们不需要在门层面实现它
 - 我们接下来实现一个超级低配版 NEMU.....

Y-Emulator (YEMU) 设计与实现

- 存储模型：内存 + 寄存器（包含PC）

- $16 + 5 = 21$ bytes = 168 bits
- 总共有 2^{168} 种不同的取值状态

- 任给一个状态，我们都能计算出PC处的指令，从而计算出下一个状态

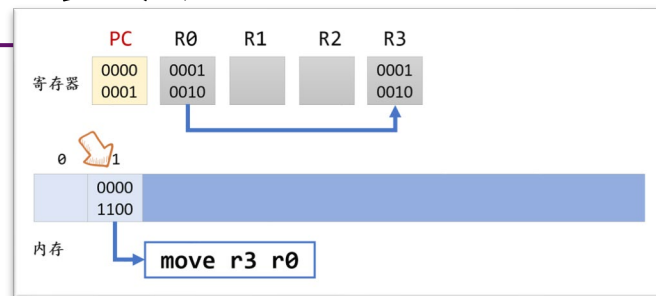


Y-Emulator (YEMU) 设计与实现

- 存储模型：内存 + 寄存器（包含PC）

- $16 + 5 = 21$ bytes = 168 bits
- 总共有 2^{168} 种不同的取值状态

- 任给一个状态，我们都能计算出PC处的指令，从而计算出下一个状态



- 理论上，任何计算机系统都是这样的状态机

- (M, R) 构成计算机的状态
- 32GiB内存有 $2^{274877906944}$ 种不同的状态 (==)
- $32 \times 1024 \times 1024 \times 1024 \times 8$ bits

- 每一个时钟周期，取出 $M[R[PC]]$ 的指令；执行；写回

- 受制于物理实现 (和功耗) 的限制，通常每个时钟周期只能改变少量寄存器和内存的状态
- 未来：(量子计算机颠覆了这个模型：同一时刻可以处于多个状态)

YEMU: 模拟存储

- **存储**是计算机能实现 “计算” 的重要基础
 - 寄存器 (PC)、内存
 - 这简单, 用全局变量就好了!

```
#include <stdint.h>
#define NREG 4
#define NMEM 16
typedef uint8_t u8; // 没用过 uint8_t?
u8 pc = 0, R[NREG], M[NMEM] = { ... };
```

- 建议 STFW (C 标准库) → bool 有没有?
- 现代计算机系统: `uint8_t == unsigned char`
 - C Tips: 使用 `unsigned int` 避免潜在的 UB
 - `-fwrapv` 可以强制有符号整数溢出为 wraparound
 - C Quiz: 把指针转换成整数, 应该用什么类型? `(int) &p`? `intptr_t`?

提升代码质量

- 还有更好地写法么?

```
#include <stdint.h>
#define NREG 4
#define NMEM 16
typedef uint8_t u8; // 没用过 uint8_t?
u8 pc = 0, R[NREG], M[NMEM] = { ... };
```


提升代码质量

- 给寄存器名字?

```
#define NREG 4
u8 R[NREG], pc; // 有些指令是用寄存器名描述的
#define RA 1 // BUG: 数组下标从0开始
...
```

```
enum { RA, R1, ..., PC };
u8 R[] = {
    [RA] = 0, // 这是什么语法? ?
    [R1] = 0,
    ...
    [PC] = init_pc,
};
```

```
enum { RA, R1, ..., PC };
u8 R[] = { 0, 0, 0, 0, 0, ..., init_pc};
```

```
#define pc (R[PC]) // 把 PC 也作为寄存器的一部分
#define NREG (sizeof(R) / sizeof(u8))
```

从一小段代码看软件设计

- 软件里有很多隐藏的 dependencies (一些额外的、代码中没有体现和约束的 “规则”)
 - 一处改了, 另一处忘了 (例如加了一个寄存器忘记更新 NREG...)
 - 减少 dependencies → 降低代码耦合程度

```
// breaks when adding a register
#define NREG 5 // 隐藏假设max{RA, RB, ... PC} == (NREG - 1)

// breaks when changing register size
#define NREG (sizeof(R) / sizeof(u8)) // 隐藏假设寄存器是8-bit

// never breaks
#define NREG (sizeof(R) / sizeof(R[0])) // 但需要R的定义

// even better (why?)
enum { RA, ... , PC, NREG }
```

PA框架代码中的CPU_state

```
why@why-VirtualBox: ~/Documents/ICS2021/ics2021/nemu/

" Press ? for help
.. (up a dir)
</ICS2021/ics2021/nemu/
> build/
> configs/
> include/
> resource/
> scripts/
> src/
>   cpu/
>   device/
>   engine/
>   isa/
>     riscv32/
>       difftest/
>       include/
>         isa-all-instr.h
>         isa-def.h

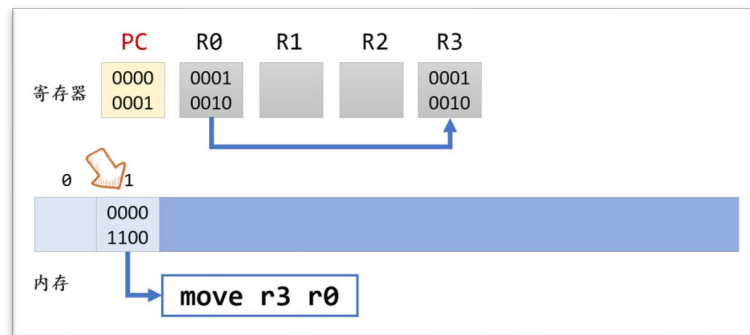
1 #ifndef __ISA_RISCV32_H
2 #define __ISA_RISCV32_H
3
4 #include <common.h>
5
6 typedef struct {
7     struct {
8         rtlreg_t _32;
9     } gpr[32];
10
11     vaddr_t pc;
12     riscv32_CPU_state;
13
14     // decode
15     typedef struct {
16         union {
17             struct {
18                 uint32_t opcode1_0 : 2;
19                 uint32_t opcode6_2 : 5;
20             };
21         };
22     };
23 } riscv32_CPU_state;
24
25 #endif
26
27 #ifndef RISCV32_REG_H
28 #define RISCV32_REG_H
29
30 #include <common.h>
31
32 static inline int check_reg_idx(int idx) {
33     IFDEF(CONFIG_RT_CHECK, assert(idx >= 0 && idx < 32));
34     return idx;
35 }
36
37 #define gpr(idx) (cpu.gpr[check_reg_idx(idx)]._32)
38
39 static inline const char* reg_name(int idx, int width) {
40     extern const char* regs[];
41     return regs[check_reg_idx(idx)];
42 }
43
44 #endif
```

- 对于复杂的情况，struct/union 是更好的设计
 - 担心性能 (check_reg_idx)?
 - 在超强的编译器优化面前，不存在的

YEMU

- 在时钟信号驱动下，根据(M, R)更新系统的状态
- RISC 处理器 (以及实际的 CISC 处理器实现):
 - 取指令 (fetch): 读出 $M[R[PC]]$ 的一条指令
 - 译码 (decode): 根据指令集规范解析指令的语义 (顺便取出操作数)
 - 执行 (execute): 执行指令、运算后写回寄存器或内存
- 最重要的就是实现 `idex()`
 - 这就是 PA 里你们最挣扎的地方 (囊括了整个手册)

```
int main() {  
    while (!is_halt(M[pc])) {  
        idex();  
    }  
}
```



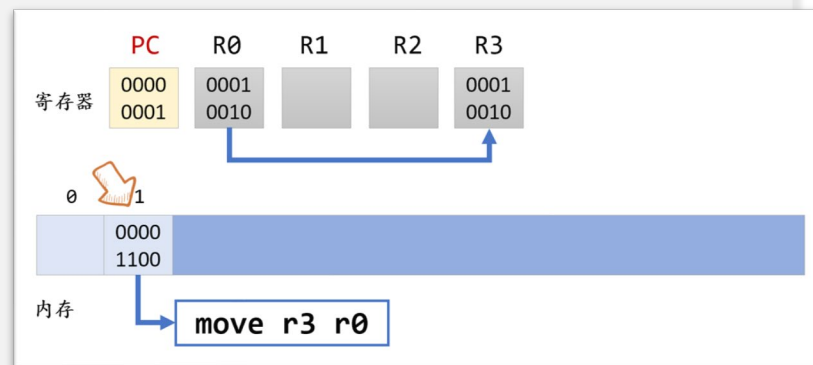
代码例子1

寄存器: PC, R0 (RA), R1, R2, R3 (8-bit)

内存: 16字节 (按字节访问)

	7	6	5	4	3	2	1	0
mov	[0	0	0	0]	[rt]	[rs]		
add	[0	0	0	1]	[rt]	[rs]		
load	[1	1	1	0]	[addr			
store	[1	1	1	1]	[addr			

```
void idex() {  
    if ((M[pc] >> 4) == 0) {  
        R[(M[pc] >> 2) & 3] = R[M[pc] & 3];  
        pc++;  
    } else if ((M[pc] >> 4) == 1) {  
        R[(M[pc] >> 2) & 3] += R[M[pc] & 3];  
        pc++;  
    } else if ((M[pc] >> 4) == 14) {  
        R[0] = M[M[pc] & 0xf];  
        pc++;  
    } else if ((M[pc] >> 4) == 15) {  
        M[M[pc] & 0xf] = R[0];  
        pc++;  
    }  
}
```



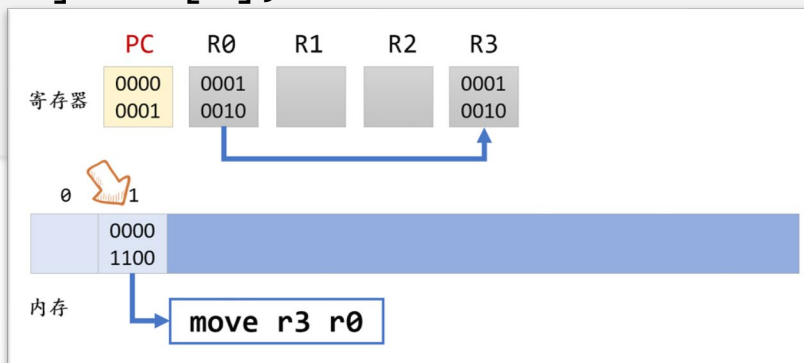
代码例子2

寄存器: PC, R0 (RA), R1, R2, R3 (8-bit)

内存: 16字节 (按字节访问)

	7	6	5	4	3	2	1	0
mov	[0	0	0	0]	[	rt]	[	rs]
add	[0	0	0	1]	[	rt]	[	rs]
load	[1	1	1	0]	[	addr	]	
store	[1	1	1	1]	[	addr	]	

```
void idex() {  
    u8 inst = M[pc++];  
    u8 op = inst >> 4;  
    if (op == 0x0 || op == 0x1) {  
        int rt = (inst >> 2) & 3, rs = (inst & 3);  
        if (op == 0x0) R[rt] = R[rs];  
        else if (op == 0x1) R[rt] += R[rs];  
    }  
    if (op == 0xe || op == 0xf) {  
        int addr = inst & 0xf;  
        if (op == 0xe) R[0] = M[addr];  
        else if (op == 0xf) M[addr] = R[0];  
    }  
}
```



代码例子3 (YEMU代码)

```
typedef union inst {  
    struct { u8 rs : 2, rt: 2, op: 4; } rtype;  
    struct { u8 addr: 4, op: 4; } mtype;  
} inst_t;
```

```
#define RTYPE(i) u8 rt = (i)->rtype.rt, rs = (i)->rtype.rs;  
#define MTYPE(i) u8 addr = (i)->mtype.addr;
```

```
void idex() {  
    inst_t *cur = (inst_t *)&M[pc];  
    switch (cur->rtype.op) {  
        case 0b0000: { RTYPE(cur); R[rt] = R[rs]; pc++; break; }  
        case 0b0001: { RTYPE(cur); R[rt] += R[rs]; pc++; break; }  
        case 0b1110: { MTYPE(cur); R[RA] = M[addr]; pc++; break; }  
        case 0b1111: { MTYPE(cur); M[addr] = R[RA]; pc++; break; }  
        default: panic("invalid instruction at PC = %x", pc);  
    }  
}
```

	7	6	5	4	3	2	1	0
mov	[0	0	0	0]	[	rt]	[	rs]
add	[0	0	0	1]	[	rt]	[	rs]
load	[1	1	1	0]	[	addr		]
store	[1	1	1	1]	[	addr		]

op	rt	rs
op	addr	

有用的C语言特性

- Union / bit fields

```
typedef union inst {  
    struct { u8 rs : 2, rt: 2, op: 4; } rtype;  
    struct { u8 addr: 4, op: 4; } mtype;  
} inst_t;
```

- 指针

- 内存只是个字节序列
- 无论何种类型的指针都只是地址 + 对指向内存的解读

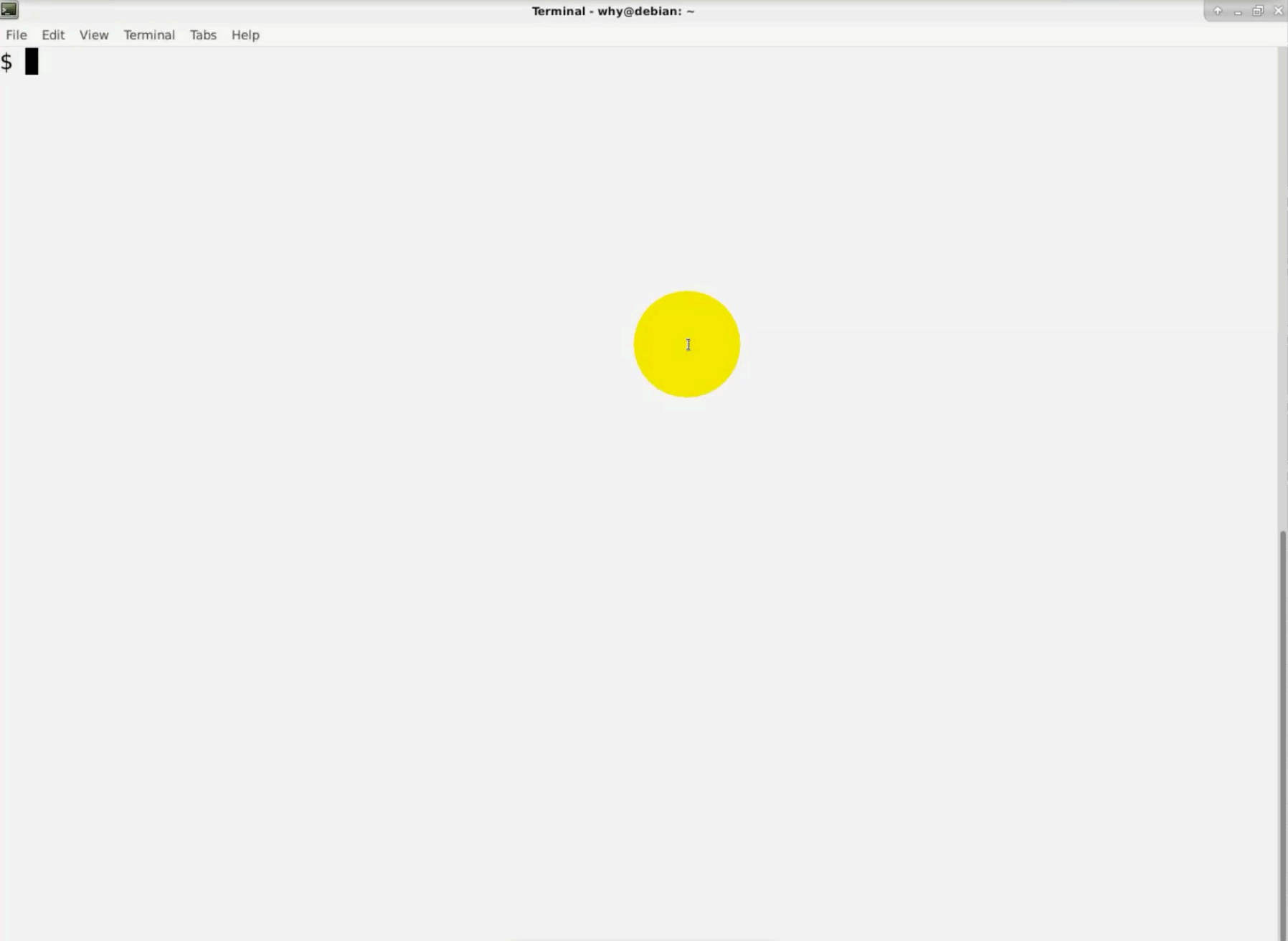
```
inst_t *cur = (inst_t *)&M[pc];  
// cur -> rtype.op  
// cur -> mtype.addr
```

```
14 // decode  
15 typedef struct {  
16     union {  
17         struct {  
18             uint32_t opcode1_0 : 2;  
19             uint32_t opcode6_2 : 5;  
20             uint32_t rd : 5;  
21             uint32_t funct3 : 3;  
22             uint32_t rs1 : 5;  
23             int32_t simm11_0 : 12;  
24         } i;  
25         struct {  
26             uint32_t opcode1_0 : 2;  
27             uint32_t opcode6_2 : 5;  
28             uint32_t imm4_0 : 5;  
29             uint32_t funct3 : 3;  
30             uint32_t rs1 : 5;  
31             uint32_t rs2 : 5;  
32             int32_t simm11_5 : 7;  
33         } s;  
34         struct {  
35             uint32_t opcode1_0 : 2;  
36             uint32_t opcode6_2 : 5;  
37             uint32_t rd : 5;  
38             uint32_t imm31_12 : 20;  
39         } u;  
40         uint32_t val;  
41     } instr;  
42 } riscv32_ISADecodeInfo;  
43
```



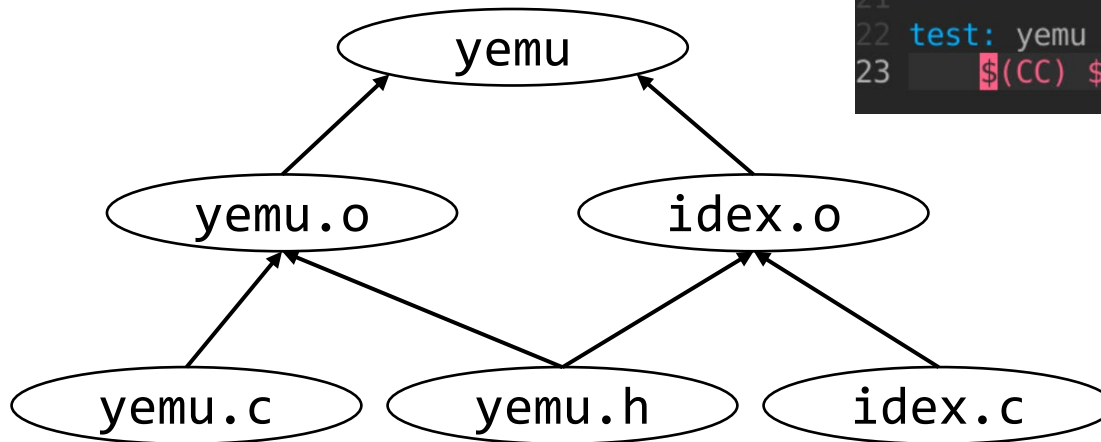
```
// --- pattern matching wrappers for decode ---
#define INSTPAT(pattern, ...) do { \
    uint64_t key, mask, shift; \
    pattern_decode(pattern, STRLEN(pattern), &key, &mask, &shift); \
    if (((INSTPAT_INST(s) >> shift) & mask) == key) { \
        INSTPAT_MATCH(s, ##__VA_ARGS__); \
        goto *(__instpat_end); \
    } \
} while (0)
```

```
65     INSTPAT("??????? ???? ???? ??? ????? 00101 11", auipc    , U, R(dest) = src1 + s->pc);
66     INSTPAT("??????? ???? ???? 011 ????? 00000 11", ld       , I, R(dest) = Mr(src1 + src2, 8));
67     INSTPAT("??????? ???? ???? 011 ????? 01000 11", sd       , S, Mw(src1 + dest, 8, src2));
68
```



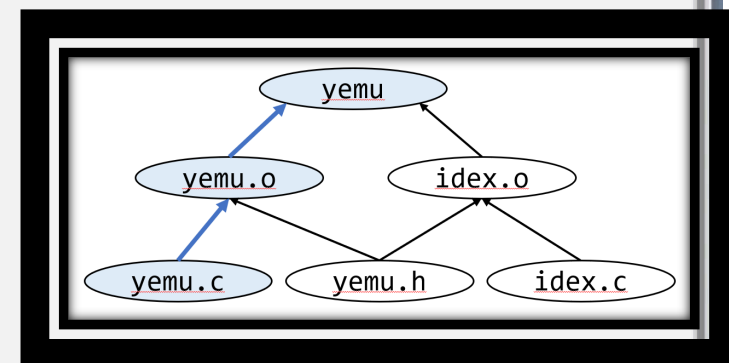
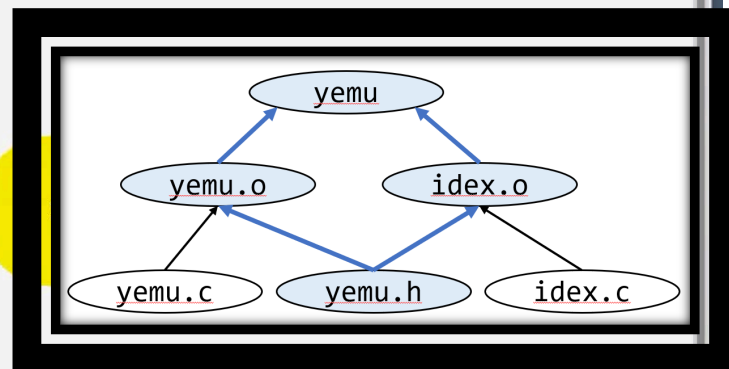
Makefile

- make



```
1 .PHONY: run clean test
2
3 CFLAGS = -Wall -Werror -std=c11 -O2
4 CC = gcc
5 LD = gcc
6
7 yemu: yemu.o idex.o
8     $(LD) $(LDFLAGS) -o yemu yemu.o idex.o
9
10 yemu.o: yemu.c yemu.h
11     $(CC) $(CFLAGS) -c -o yemu.o yemu.c
12
13 idex.o: idex.c yemu.h
14     $(CC) $(CFLAGS) -c -o idex.o idex.c
15
16 run: yemu
17     @./yemu
18
19 clean:
20     rm -f test yemu *.o
21
22 test: yemu
23     $(CC) $(CFLAGS) -o test idex.o test.c && ./test
```

```
Applications Terminal - why@debian... 16:30 EN why  
Terminal - why@debian: ~  
File Edit View Terminal Tabs Help  
$ cd ICS_teach/  
$ ls  
yemu yemu.tar.gz  
$ cd yemu/  
$ ls  
idex.c idex.o Makefile test.c yemu yemu.c yemu.h yemu.o  
$ make clean  
rm -f test yemu *.o  
$
```



```
$ make run
```

```
Hit GOOD trap @ pc = 5.
```

```
M[00] = 0xe7 (231)
```

```
M[01] = 0x04 (4)
```

```
M[02] = 0xe6 (230)
```

```
M[03] = 0x11 (17)
```

```
M[04] = 0xf8 (248)
```

```
M[05] = 0x00 (0)
```

```
M[06] = 0x10 (16)
```

```
M[07] = 0x21 (33)
```

```
M[08] = 0x31 (49)
```

```
M[09] = 0x00 (0)
```

```
M[10] = 0x00 (0)
```

```
M[11] = 0x00 (0)
```

```
M[12] = 0x00 (0)
```

```
M[13] = 0x00 (0)
```

```
M[14] = 0x00 (0)
```

```
M[15] = 0x00 (0)
```

```
$ vim Makefile
```

```
$ make test
```

```
gcc -Wall -Werror -std=c11 -O2 -o test idx.o test.c && ./test
```

```
Test #1 (0b11100111): PASS
```

```
Test #2 (0b000000100): PASS
```

```
Test #3 (0b11100101): PASS
```

```
Test #4 (0b000010001): PASS
```

```
Test #5 (0b11110111): PASS
```

```
1 .PHONY: run clean test
2
3 CFLAGS = -Wall -Werror -std=c11 -O2
4 CC = gcc
5 LD = gcc
6
7 yemu: yemu.o idx.o
8     $(LD) $(LDFLAGS) -o yemu yemu.o idx.o
9
10 yemu.o: yemu.c yemu.h
11     $(CC) $(CFLAGS) -c -o yemu.o yemu.c
12
13 idx.o: idx.c yemu.h
14     $(CC) $(CFLAGS) -c -o idx.o idx.c
15
16 run: yemu
17     @./yemu
18
19 clean:
20     rm -f test yemu *.o
21
22 test: yemu
23     $(CC) $(CFLAGS) -o test idx.o test.c && ./test
```

make test

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 #include "yemu.h"
5
6 u8 R[NREG], M[NMEM], oldR[NREG], oldM[NMEM], *newM = M, *newR = R;
7
8 void random_rm() {
9     for (int i = 0; i < NREG; i++)
10         R[i] = rand() & 0xff;
11     for (int i = 0; i < NMEM; i++)
12         M[i] = rand() & 0xff;
13 }
14
15 #define ASSERT_EQ(a, b) ((u8)(a) == (u8)(b))
16
17 #define TESTCASES(_) \
18     _(1, 0b11100111, random_rm, ASSERT_EQ(newR[0], oldM[7])) \
19     _(2, 0b00000100, random_rm, ASSERT_EQ(newR[1], oldR[0])) \
20     _(3, 0b11100101, random_rm, ASSERT_EQ(newR[0], oldM[5])) \
21     _(4, 0b00010001, random_rm, ASSERT_EQ(newR[0], oldR[0] + oldR[1])) \
22     _(5, 0b11110111, random_rm, ASSERT_EQ(newM[7], oldR[0])) \
23
24 #define MAKETEST(id, inst, precondition, postcond) { \
25     precondition; \
26     memcpy(oldM, M, NMEM); \
27     memcpy(oldR, R, NREG); \
28     pc = 0; M[pc] = inst; \
29     idex(); \
30     printf("Test #%d (%s): %s\n", \
31         id, #inst, (postcond) ? "PASS" : "FAIL"); \
32 }
33
34 int main() {
35     TESTCASES(MAKETEST)
36     return 0;
37 }
```

预编译的解析

- gcc -E test.c | indent - | vim -

```
#34 "test.c"
int
main ()
{
  {
    random_rm ();
    memcpy (oldM, M, 16);
    memcpy (oldR, R, NREG);
    (R[PC]) = 0;
    M[(R[PC])] = 0 b11100111;
    idex ();
    printf ("Test #%d (%s): %s\n", 1, "0b11100111",
           (((u8) (newR[0]) == (u8) (oldM[7]))) ? "PASS" : "FAIL");
  }
  {
    random_rm ();
    memcpy (oldM, M, 16);
    memcpy (oldR, R, NREG);
    (R[PC]) = 0;
    M[(R[PC])] = 0 b00000100;
    idex ();
    printf ("Test #%d (%s): %s\n", 2, "0b00000100",
           (((u8) (newR[1]) == (u8) (oldR[0]))) ? "PASS" : "FAIL");
  }
  {
    random_rm ();
    memcpy (oldM, M, 16);
    memcpy (oldR, R, NREG);
    (R[PC]) = 0;
    M[(R[PC])] = 0 b11100101;
    idex ();
    printf ("Test #%d (%s): %s\n", 3, "0b11100101",
           (((u8) (newR[0]) == (u8) (oldM[5]))) ? "PASS" : "FAIL");
  }
}
```

小结

- 如何管理 “更大” 的项目 (YEMU)?
 - 我们分多个文件管理它
 - `yemu.h` - 寄存器名; 必要的声明
 - `yemu.c` - 数据定义、主函数
 - `idex.c` - 译码执行
 - `Makefile` - 编译脚本 (能实现增量编译)
- 使用合理的编程模式
 - 减少模块之间的依赖
 - `enum { RA, ... , NREG }`
 - 合理使用语言特性, 编写可读、可证明的代码
 - `inst_t *cur = (inst_t *)&M[pc]`
- NEMU 就是加强版的 YEMU

更多的计算机系统模拟器

- `am-kernels/litenes`
 - 一个“最小”的 NES 模拟器
 - 自带硬编码的 ROM 文件
- `fceux-am`
 - 一个非常完整的高性能 NES 模拟器
 - 包含对卡带定制芯片的模拟 (`src/boards`)
- QEMU
 - 工业级的全系统模拟器
 - 2011 年发布 1.0 版本
 - 有兴趣的同学可以 RTFSC
 - 作者：传奇黑客 Fabrice Bellard

End.

永远不要停止对好代码的追求

用红白机模拟器NES Emulator玩Mario

